WebSphere® Application Server - Express for IBM i, Version 7.0

Administering WebSphere applications

Note

Before using this information, be sure to read the general information under “Notices” on page 2399.

Compilation date: September 24, 2008

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Changes to serve you more quickly

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- Links to Web addresses beginning with http:// work
- Links that refer to specific page numbers within the same PDF book work
- The remaining links will *not* work. You receive an error message when you click them

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Web applications

Tuning URL invocation cache

The URL invocation cache holds information for mapping request URLs to servlet resources. A cache of the requested size is created for each worker thread that is available to process a request. The default size of the invocation cache is 50. If more than 50 unique URLs are actively being used (each JavaServer Page is a unique URL), you should increase the size of the invocation cache.

Before you begin

A larger cache uses more of the Java™ heap, so you might also need to increase the maximum Java heap size. For example, if each cache entry requires 2KB, maximum thread size is set to 25, and the URL invocation cache size is 100; then 5MB of Java heap are required.

The invocation cache is now Web container based instead of thread-based, and shared for all Web container threads.

About this task

To change the size of the invocation cache:

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** and select the application server that you are tuning.
2. Click **Java and Process Management**.
3. Click **Process Definition** under Additional Properties.
4. Click **Java Virtual Machine** under Additional Properties.
5. Click **Custom Properties** under Additional Properties.
6. Specify **invocationCacheSize** in the Name field and the size of the cache in the Value field. The default size for the invocation cache is 500 entries. Since the invocation cache is no longer thread-based, the invocation cache size specified by the user is multiplied by ten to provide similar function from previous releases. For example, if you specify an invocation cache size of 50, the Web container will create a cache size of 500.
7. Click **Apply** and then **Save** to save your changes.
8. Stop and restart the application server.

Results

The new cache size is used for the URL invocation cache.

Developing servlet applications using asynchronous request dispatcher

Web modules can dispatch requests concurrently on separate threads. Requests can be dispatched by the server or client.

Before you begin

For additional information about the AsyncRequestDispatcherConfig and the AsyncRequestDispatcher interfaces, review the com.ibm.websphere.webcontainer.async package in the application programming interfaces (API) documentation. The generated API documentation is available in the information center table of contents from the path Reference > APIs - Application Programming Interfaces.

Review the asynchronous request dispatcher application (ARD) design considerations topic before completing the following steps.

About this task

Concurrent dispatching can improve servlet response time. If operations are dependant on each other, do not enable asynchronous request dispatching, therefore, select Disabled. Concurrent dispatching might result in errors when operations are dependant. Select Server side to enable the server to aggregate requests dispatched concurrently. Select Client side to enable the client to aggregate requests dispatched concurrently.

1. Logically separate resource intensive operations.
2. Develop servlets that use an asynchronous request dispatcher to include these operations.
3. Enable asynchronous request dispatching on an application server.
4. Deploy the application in an application server that has asynchronous request dispatching enabled.
5. Select an aggregation type for the application that needs ARD.
6. Optional: Configure the AsyncRequestDispatcherWorkManager work manager that is used for the request dispatch threads.
7. Restart the application server.

What to do next

Restart the modified applications if already installed or start newly installed applications to enable ARD on each application.

Asynchronous request dispatcher

Asynchronous request dispatcher (ARD) can improve Servlet response time when slow operations can be logically separated and performed concurrently with other operations required to complete the response. ARD enables Java™ servlet programmers to perform standard javax.servlet.RequestDispatcher include calls for the same request concurrently on separate threads. These javax.servlet.RequestDispatcher include calls are completed sequentially on the same thread. ARD is also useful in low CPU, long wait situations like waiting for a database connection.

If there are large CPU or memory requirements, ARD alone does not alleviate those issues. However, in combination with the remote request dispatcher, operations driven by one servlet request that can be performed concurrently on multiple application servers, alleviating resource demand on a single server and decreasing the risk of a system down situation.

Servlets, portlets, and JavaServer Pages (JSP) files can all utilize ARD. This functionality is an extension beyond the requirements of the Java Servlet Specification , which only describes synchronous request dispatching. ARD requires a new channel, called the ARD channel, between the HTTP and Web container channels to form a new channel chain. These new chains correspond only to the existing default host chains and reuse the same ports.

Each include can write output to the client and because ordering is important for valid results, there must be some aggregation of the data written. Typically, a servlet writes data to a buffer and once full, it is flushed to client. For server-side aggregation, the ARD channel cannot flush until any includes that had placeholders written to the current buffer are finished.

Client-side aggregation of the asynchronous include is also supported. Web 2.0 programmers often use Asynchronous JavaScript™ and XML (Ajax) in the Web browser of the client to dynamically retrieve and aggregate remote resources. Unfortunately, this puts the burden on the programmer to aggregate the contents and learn new technologies. Client-side aggregation automatically adds the necessary JavaScript

to dynamically update the page. For non-JavaScript clients, you can switch ARD to server-side aggregation, which gives equivalent results. You can deny non-JavaScript clients when using client-side aggregation.

ARD uses the Web container APIs to plug in unique request dispatching logic. It interacts with WCCM to read in configuration information for enablement status per enterprise application as well as a global appserver setting. You can use the administrative console and wsAdmin to enable or disable ARD. Servlets, portlets, and JSP files can all utilize ARD.

Related tasks

"Developing servlet applications using asynchronous request dispatcher" on page 1
Web modules can dispatch requests concurrently on separate threads. Requests can be dispatched by the server or client.

Related reference

"Asynchronous request dispatcher application design considerations"
Asynchronous request dispatcher (ARD) is not a one-size-fits-all solution to servlet programming. You must evaluate the needs of your application and the caveats of using ARD. Switching all includes to start asynchronously is not the solution for every scenario, but when used wisely, ARD can increase response time. This article contains important details about the ARD implementation and issues to consider when you design an application that leverages ARD.

Related information

Asynchronous request dispatcher application design considerations

Asynchronous request dispatcher (ARD) is not a one-size-fits-all solution to servlet programming. You must evaluate the needs of your application and the caveats of using ARD. Switching all includes to start asynchronously is not the solution for every scenario, but when used wisely, ARD can increase response time. This article contains important details about the ARD implementation and issues to consider when you design an application that leverages ARD.

Asynchronous request dispatcher client-side implementation

- JavaScript is dynamically written to the response output.
- This JavaScript results in Ajax requests back to a server-side results provider.
- Because of the Asynchronous Input/Output (AIO) features of the channel, the Ajax request does not tie up a thread and instead is notified for completion through an include callback.
- The client only makes one request at a time for the asynchronous includes because of browser limitations in the number of connections.
- Original connection has to be valid for the lifetime of the includes. It cannot be reused for the Ajax requests.
- Comment nodes, such as following,

  ```
  <!--uniquePlaceholderID--><!--1-->
  ```

  are placed in the browser object model since comment nodes have no effect on the page layout.
- Whenever a complete fragment exists, a response can be sent to the client and the comment node with the same ID is replaced. Requests are made until all the fragments are retrieved.
- Verify applications on all supported browsers when using client-side aggregation. Object oriented JavaScript principles are used so that applications only need avoid using the method name getDynamicDataIBMARD. Any previously specified window.onload is started before the ARD onload method.

Asynchronous request dispatcher channel results service

Requests for include data from the asynchronous JavaScript code are sent to known Uniform Resource Identifiers, URIs also known as URLs, that the ARD channel can intercept to prevent traveling through Web container request handling. These URIs are unique for the each server restart.

For example, `/IBMARD01234567/asyncInclude.js` is the URI for the JavaScript that forces the retrieval of the results, and `/IBMARD01234567/IBMARDQueryStringEntries?=12000` is used to retrieve the results for the entry with ID 12000.

To prevent unauthorized results access, unique IDs are generated for the service URI and for the ARD entries. A common ID generator is shared among the session and ARD, so uniqueness is configurable through session configuration. Session IDs are considered secure, but they are not as secure as using a Lightweight Third-Party Authentication (LTPA) token.

Custom client-side aggregation

If you want to perform your own client-side aggregation, the isUseDefaultJavascript method must return as false. The isUseDefaultJavascript method is part of the AsyncRequestDispatcherConfig method, which is set on the AsyncRequestDispatcher or for the AsyncRequestDispatcherConfigImpl.getRef method. The AsyncRequestDispatcherConfigImpl.getRef method is the global configuration object. You might want to perform your own client-side aggregation if the back button functionality is problematic. You must remove the results from the generic results service to prevent memory leaks, so that multiple requests with the same response results through an XMLHttpRequest fail. To facilitate proper location of position, placeholders are still written in the code as

```
<!--uniquePlaceholderID--><!--x-->
```

where x is the order of the includes. The endpoint to retrieve results are retrieved from the request attribute com.ibm.websphere.webcontainer.ard.endpointURI.

When making a request to the endpoint, ARD sends as many response fragments as possible when the request is made. Therefore, the client needs to re-request if all fragments are not initially returned. Trying to display the results directly in a browser without using an XMLHttpRequest can result in errors related to non well-formed XML. The response data is returned in the following format with a content type of text/xml:

```
<div id="2"><BR>Servlet 3--dispatcher3 requesting Servlet3 to sleep for 0 seconds at: 1187967704265
<BR> Servlet 3--Okay, all done!  This should print pop up: third at: 1187967704281 </div>
```

For additional information about the AsyncRequestDispatcherConfig and the AsyncRequestDispatcher interfaces, review the com.ibm.websphere.webcontainer.async package in the application programming interfaces (API) documentation. The generated API documentation is available in the information center table of contents from the path **Reference** → **APIs - Application Programming Interfaces**.

Server-side aggregation

Like client-side aggregation, server-side aggregation uses the ARD channel as a results service. The ARD channel knows which asynchronous includes have occurred for certain set of buffers. Those buffers can then be searched for an include placeholder. Because of the issues of JSP buffering, the placeholder for the include might not be in the searched buffers. If this occurs, the next set of buffers must also look for any include placeholders missed in the previous set. ARD attempts to iteratively aggregate as includes return so that response content can be sent to the client as soon as possible.

Asynchronous beans

An AsynchBeans work manager is used to start the includes. If the number of currently requested includes is greater than the work manager maximum thread pool size and this size is not growable, it starts the work on the current thread and skips the placeholder write. Utilizing AsynchBeans suppports propagation

of the J2EE context of the original thread including work area, internationalization, application profile, z/OS® operation system work load management, security, transaction, and connection context.

Timer

A single timer is used for ARD and timer tasks are created for all the timeout types of ARD requests. Tasks registered with the timer are not guaranteed to run at the exact time specified because the timer runs on a single thread, therefore one timeout might have to wait for the other timeout actions to complete. The timer is used as a last resort.

Remote request dispatcher

Optionally, ARD can be used in concert with the remote request dispatcher. The remote request dispatcher was introduced in WebSphere® Application Server Network Deployment 6.1. The remote request dispatcher runs the include on a different application server in a core group by serializing the request context into a SOAP message and using Web services to call the remote server. This is useful when the expense of creating and sending a SOAP message through Web services is outweighed by issuing the request locally. For more information, see this developerWorks article.

Exceptions

In the case of an exception in an included servlet, the Web container goes through the error page definitions mapped to exception types. So an error page defined in the deployment descriptor shows up as a portion of the aggregated page. Insert logic into the error page itself if behavior is different for an include. Because the include runs asynchronously, there is no guarantee that the top level servlet is still in service, therefore the exception is not propagated back from an asynchronous include like a normal include. Other includes finish so that partial pages can be displayed.

If the ARD work manager runs out of worker threads, the include is processed like a synchronous include. This is the default setting, but the work manager can also grow such that it does not result in this condition. This change in processing is invisible to the user during processing but is noted once in the system logs as a warning message and the rest of the time in the trace logs when enabled. Other states that can trigger the include to occur synchronously are reaching the maximum percentage of expired requests over a time interval and reaching the maximum size of the results store.

There are cases where exceptions happen outside of the scope of normal error page handling. For example, work can be rejected by the work manager. A timer can expire waiting for an include response to return. The ARD channel, acting as a generic service to retrieve the results, might receive an ID that is not valid. In these cases, there is no path to the error page handling because the context is missing, such as ServletRequest, ServletResponse, and ServletContext, for the request to work. To mitigate these issues, you can use the AsyncRequestDispatcherConfig interface to provide custom error messages. Defaults are provided and internationalized as needed.

Exceptions can also occur outside the scope of the request the custom configuration was set on, such as on the subsequent client-side XMLHttpRequests. In this case, the global configuration must be altered. This can be retrieved through com.ibm.wsspi.ard.AsyncRequestDispatcherConfigImpl.getRef().

Include start
: The work manager provides a timeout for how long to wait for an include to start. Since this typically happens immediately, there is not a programmatic way to enable this. However, this is configurable in the work manager settings. By default, you will not encounter this because of the maximum thread check before scheduling the work. Work can be retried if setRetriable(true) is called on the in use AsyncRequestDispatcherConfig.

Include finish
: The initiated timeout starts after the work is accepted. It can be configured through the console or programmatically through the AsyncRequestDispatcherConfig.setExecutionTimeoutOverride

method; The default value is 60000 ms, or one minute. In place of the include results, the message from the AsyncRequestDispatcherConfig.setExecutionTimeoutMessage is sent. If this intiated timeout is reached, but the actual include results are ready when the data can be flushed, preference is given to the actual results. Also, this does not apply to insertFragmentBlocking calls which always wait until the include is completed.

Expiration of results

Since the client-side has to hold the results in a service to send for the Ajax request, we want a way to expire the results if the client goes down and never retrieves the entry. The default of a minute is sufficient for a typical request because the Ajax request would come in immediately after sending the response. The timer can be configured programmatically via the setExpirationTimeoutOverride method the of AsyncRequestDispatcherConfig. The message from the getOutputRetrievalFailureMessage method of AsyncRequestDispatcherConfig is displayed when someone tries to access an entry that has expired and been removed from cache. This message is the same message that is sent to someone requesting a result with an ID that never existed.

Includes versus fragments

Consider which operations can be done asynchronously and when they can start. Ideally, all the includes are completed when the getFragment calls are made at the beginning of the request so that the includes can have more time to complete, and upon inserting the fragments, there would be less extra buffering and aggregating if they have completed. However, simply calling an asynchronous include is easier because it follows the same pattern as a normal request dispatcher include.

Web container

ServletContext

When doing cross-context includes, the context that is a target of the include must also have ARD enabled because the Web application must have been initialized for ARD for its servlet context to have valid methods to retrieve an AsyncRequestDispatcher. The aggregation type is determined by the original context's configuration because you cannot mix aggregation types.

ServletRequest

You must clone the request for each include. Otherwise, conflicts between threads might occur. Because applications can wrap the default request objects, your wrappers must implement the com.ibm.wsspi.webcontainer.servlet.IServletRequest interface, which has one method, the public Object clone method, which creates the CloneNotSupportedException.

Unwrapping occurs until a request wrapper that implements this interface is found. Non-implementing wrappers are lost; however, a servlet filter configured for the include can rewrap the response.

Changes made to the ServletRequest are not propagated back to the top level servlet unless transferState on the AsyncRequestDispatcherConfig is enabled and insertFragmentBlocking is called.

ServletResponse

A wrapped response extending com.ibm.websphere.servlet.response.StoredResponse is created by ARD and sent to the includes because the response output must be retrievable beyond the lifecycle of the original response.

Internal headers set in asynchronous includes are not supported due to lifecycle restrictions unless transferState on the AsyncRequestDispatcher config is enabled and insertFragmentBlocking is called. Normal headers are not supported in a synchronous include as specified by the servlet specification.

Include filters can rewrap the new response and must flush upon completion.

ServletInputStream
An application reading parameters using getParameter is not problematic.Parsing of parameters is forced before the first asynchronous include to prevent concurrent access to the input stream.

HttpSession
Initial getSession calls that result in a Set-Cookie header must be called from the top level servlet because it is unpredictable when the includes are started and if the headers have already been flushed. The exception is when transferState on the AsyncRequestDispatcherConfig is enabled and an insertFragmentBlocking is called. This normally creates an exception when you add the header.

Filters If there is a filter for an include, the filter is issued on the asynchronous thread.

Nested asynchronous includes
Nested asynchronous includes are not supported because they complicate aggregation. However, an asynchronous include can have nested synchronous includes. Any attempt to perform a nested asynchronous include reverts back to a synchronous include.

Transactions

Every asynchronous bean method is called using its own transaction, much like container-managed transactions in typical enterprise beans. The runtime starts a local transaction before invoking the method. The asynchronous bean method can start its own global transaction if this transaction is possible for the calling J2EE component.

If the asynchronous bean method creates an exception, any local transactions are rolled back. If the method returns normally, any incomplete local transactions are completed according to the unresolved action policy configured for the bean. If the asynchronous bean method starts its own global transaction and does not commit this global transaction, the transaction is rolled back when the method returns.

Connection management

An asynchronous bean method can use the connections that its creating servlet obtained using java:comp resource references. However, the bean method must access those connections using a get, use or close pattern. There is no connection caching between method calls on an asynchronous bean. The connection factories or data sources can be cached, but the connections must be retrieved on every method call, used, and then closed. While the asynchronous bean method can look up connection factories using a global Java Naming and Directory Interface (JNDI) name, this is not recommended for the following reasons:

- The JNDI name is hard coded in the application, for example, as a property or string literal.
- The connection factories are not shared because there is no way to specify a sharing scope.

Evaluate high load scenarios because asynchronous includes might increase the number of threads waiting on the connection.

Performance

Because includes are completed asynchronously, the total performance data for a request must take into consideration the performance of the asynchronous includes. The total time of the request could previously be understood by the time for the top level servlet to complete, but now that servlet is exiting before the includes are completed. The top level servlet still accounts for much of the additional setup time required for each include.

Therefore, a new ARD performance metric was added to the Performance Monitoring Infrastructure to measure the time for a complete request through the ARD channel. The granularity of these metrics is at the request URI level.

Since ARD is an optional feature that has to be enabled, no performance decline is seen when not utilizing ARD. However, non-ARD applications that reside on an ARD-enabled application server would suffer from the extra layer of the ARDChannel. The channel layer does not know to which application it is going so it is either on or off for all applications in a channel chain. These are defined per virtual host.

Security

Security is not invoked on synchronous include dispatches according to the servlet specification. However, security context is passed along through AsynchBeans to support programmatic usage of the isUserInRole and getUserPrincipal methods on the ServletRequest. This security context can also be propagated across to a remote request dispatch utilizing Web services security.

Related concepts

"Asynchronous request dispatcher" on page 2
Asynchronous request dispatcher (ARD) can improve Servlet response time when slow operations can be logically separated and performed concurrently with other operations required to complete the response. ARD enables Java™ servlet programmers to perform standard javax.servlet.RequestDispatcher include calls for the same request concurrently on separate threads. These javax.servlet.RequestDispatcher include calls are completed sequentially on the same thread. ARD is also useful in low CPU, long wait situations like waiting for a database connection.

Related tasks

"Developing servlet applications using asynchronous request dispatcher" on page 1
Web modules can dispatch requests concurrently on separate threads. Requests can be dispatched by the server or client.

Asynchronous request dispatching settings

Use this page to enable the asynchronous request dispatcher (ARD), which enables servlets and JSP pages to make standard include calls concurrently on separate threads.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container** → **Asynchronous request dispatching**.

Additionally, the initiation and the insertion of the include contents can be separated so that the include has more time to execute before it needs to be written to the response. ARD requires aggregation of the include contents with the original response. The application server can aggregate the contents in memory or the client browser can aggregate the contents through AJAX. Aggregation type is configurable at the application level.

Related concepts

"Asynchronous request dispatcher" on page 2
Asynchronous request dispatcher (ARD) can improve Servlet response time when slow operations can be logically separated and performed concurrently with other operations required to complete the response. ARD enables Java™ servlet programmers to perform standard javax.servlet.RequestDispatcher include calls for the same request concurrently on separate threads. These javax.servlet.RequestDispatcher include calls are completed sequentially on the same thread. ARD is also useful in low CPU, long wait situations like waiting for a database connection.

Related tasks

"Developing servlet applications using asynchronous request dispatcher" on page 1
Web modules can dispatch requests concurrently on separate threads. Requests can be dispatched by the server or client.

Allow Asynchronous Request Dispatching

Enables applications installed on this server to use asynchronous request dispatching.

Asynchronous include timeout
Specifies the default timeout in milliseconds to complete asynchronous includes.

Data type	Integer
Units	Milliseconds
Default	60000
Range	

Maximum expired requests per minute
Specifies the maximum percentage of expired response versus total response in one minute before switching to synchronous requests.

Data type	Integer
Units	Percentage
Default	15
Range	

Maximum memory size of results store
Specifies the maximum size of store for client side requests.

Data type	Integer
Units	Megabytes
Default	100
Range	

Task overview: Managing HTTP sessions

IBM® WebSphere Application Server provides a service for managing HTTP sessions, Session Manager. The key activities for session management are summarized in this topic.

About this task

Before you begin these steps, make sure you are familiar with the programming model for accessing HTTP session support in the applications following the Servlet 2.5 API.

1. Plan your session tracking approach for session management.
2. Create or modify your own applications to use session support to maintain sessions on behalf of Web applications.
3. Assemble your application.
4. Deploy your application.
5. Ensure the administrator appropriately configures session management in the administrative domain.
6. Adjust configuration settings and perform other tuning activities for optimal use of sessions in your environment.

Sessions

A session is a series of requests to a servlet, originating from the same user at the same browser.

Sessions allow applications running in a Web container to keep track of individual users.

For example, a servlet might use sessions to provide ″shopping carts″ to online shoppers. Suppose the servlet is designed to record the items each shopper indicates he or she wants to purchase from the Web site. It is important that the servlet be able to associate incoming requests with particular shoppers. Otherwise, the servlet might mistakenly add Shopper_1's choices to the cart of Shopper_2.

A servlet distinguishes users by their unique session IDs. The session ID arrives with each request. If the user's browser is cookie-enabled, the session ID is stored as a cookie. As an alternative, the session ID can be conveyed to the servlet by URL rewriting, in which the session ID is appended to the URL of the servlet or JavaServer Pages (JSP) file from which the user is making requests. For requests over HTTPS or Secure Sockets Layer (SSL), another alternative is to use SSL information to identify the session. Session tracking using the SSL ID is deprecated in WebSphere Application Server version 7.0. You can configure session tracking to use cookies or modify the application to use URL rewriting.

HTTP session migration

There are no programmatic changes required to migrate from version 5.x to version 6.x. This article describes features that are available after migration.

Migration from Version 5.x

Note: In Version 5 and later, default write frequency mode is TIME_BASED_WRITES, which is different from Version 4.0.x default mode of END_OF_SERVICE.

Session security support

You can integrate HTTP sessions and security in WebSphere Application Server. When security integration is enabled in the session management facility and a session is accessed in a protected resource, you can access that session only in protected resources from then on.

You cannot mix secured and unsecured resources accessing sessions when security integration is turned on. Security integration in the session management facility is not supported in form-based login with SWAM.

Note: SWAM is deprecated in WebSphere Application Server Version 7.0 and will be removed in a future release.

Security integration rules for HTTP sessions

Only authenticated users can access sessions created in secured pages and are created under the identity of the authenticated user. Only this authenticated user can access these sessions in other secured pages. To protect these sessions from unauthorized users, you cannot access them from an unsecured page.

Programmatic details and scenarios

WebSphere Application Server maintains the security of individual sessions.

An identity or user name, readable by the com.ibm.websphere.servlet.session.IBMSession interface, is associated with a session. An unauthenticated identity is denoted by the user name `anonymous`. WebSphere Application Server includes the com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException class, which is used when a session is requested without the necessary credentials.

The session management facility uses the WebSphere Application Server security infrastructure to determine the authenticated identity associated with a client HTTP request that either retrieves or creates a session. WebSphere Application Server security determines identity using certificates, LPTA, and other methods.

After obtaining the identity of the current request, the session management facility determines whether to return the session requested using a getSession call.

The following table lists possible scenarios in which security integration is enabled with outcomes dependent on whether the HTTP request is authenticated and whether a valid session ID and user name was passed to the session management facility.

	Unauthenticated HTTP request is used to retrieve a session	**HTTP request is authenticated, with an identity of ″FRED″ used to retrieve a session**
No session ID was passed in for this request, or the ID is for a session that is no longer valid	A new session is created. The user name is `anonymous`	A new session is created. The user name is `FRED`
A session ID for a valid session is passed in. The current session user name is ″anonymous″	The session is returned.	The session is returned. session management changes the user name to `FRED`
A session ID for a valid session is passed in. The current session user name is `FRED`	The session is not returned. An UnauthorizedSessionRequestException error is created*	The session is returned.
A session ID for a valid session is passed in. The current session user name is `BOB`	The session is not returned. An UnauthorizedSessionRequestException error is created*	The session is not returned. An UnauthorizedSessionRequestException error is created*

* A com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException error is created to the servlet.

Session management support

WebSphere Application Server provides facilities, grouped under the heading *Session Management*, that support the javax.servlet.http.HttpSession interface described in the Servlet API specification.

In accordance with the Servlet 2.3 API specification, the session management facility supports session scoping by Web modules. Only servlets in the same Web module can access the data associated with a particular session. Multiple requests from the same browser, each specifying a unique Web application, result in multiple sessions with a shared session ID. You can invalidate any of the sessions that share a session ID without affecting the other sessions.

You can configure a session timeout for each Web application. A Web application timeout value of 0 (the default value) means that the invalidation timeout value from the session management facility is used.

When an HTTP client interacts with a servlet, the state information associated with a series of client requests is represented as an HTTP session and identified by a session ID. Session management is responsible for managing HTTP sessions, providing storage for session data, allocating session IDs, and tracking the session ID associated with each client request through the use of cookies or URL rewriting techniques. Session management can store session-related information in several ways:

- In application server memory (the default). This information cannot be shared with other application servers.
- In a database. This storage option is known as *database persistent sessions*.

The last two options are referred to as *distributed sessions*. Distributed sessions are essential for using HTTP sessions for the failover facility. When an application server receives a request associated with a session ID that it currently does not have in memory, it can obtain the required session state by accessing the external store (database or memory-to-memory). If distributed session support is not enabled, an application server cannot access session information for HTTP requests that are sent to servers other than the one where the session was originally created. Session management implements caching optimizations to minimize the overhead of accessing the external store, especially when consecutive requests are routed to the same application server.

Storing session states in an external store also provides a degree of fault tolerance. If an application server goes offline, the state of its current sessions is still available in the external store. This availability enables other application servers to continue processing subsequent client requests associated with that session.

Saving session states to an external location does not completely guarantee their preservation in case of a server failure. For example, if a server fails while it is modifying the state of a session, some information is lost and subsequent processing using that session can be affected. However, this situation represents a very small period of time when there is a risk of losing session information.

The drawback to saving session states in an external store is that accessing the session state in an external location can use valuable system resources. session management can improve system performance by caching the session data at the server level. Multiple consecutive requests that are directed to the same server can find the required state data in the cache, reducing the number of times that the actual session state is accessed in external store and consequently reducing the overhead associated with external location access.

Session tracking options

HTTP session support also involves session tracking. You can use cookies, URL rewriting, or Secure Sockets Layer (SSL) information for session tracking.

the following tracking methods are available:
- Session tracking with cookies
- Session tracking with URL rewriting
- Session tracking with Secure Sockets Layer (SSL) information

Session tracking with cookies

Tracking sessions with cookies is the default. No special programming is required to track sessions with cookies.

Session tracking with URL rewriting

An application that uses URL rewriting to track sessions must adhere to certain programming guidelines. The application developer needs to do the following:
- Program servlets to encode URLs
- Supply a servlet or JavaServer Pages (JSP) file as an entry point to the application

Using URL rewriting also requires that you enable URL rewriting in the session management facility.

Note: In certain cases, clients cannot accept cookies. Therefore, you cannot use cookies as a session tracking mechanism. Applications can use URL rewriting as a substitute.

Program session servlets to encode URLs

Depending on whether the servlet is returning URLs to the browser or redirecting them, include either the encodeURL method or the encodeRedirectURL method in the servlet code. Examples demonstrating what to replace in your current servlet code follow.

Rewrite URLs to return to the browser

Suppose you currently have this statement:

```
out.println("<a href=\"/store/catalog\">catalog<a>");
```

Change the servlet to call the encodeURL method before sending the URL to the output stream:

```
out.println("<a href=\"");
out.println(response.encodeURL ("/store/catalog"));
out.println("\">catalog</a>");
```

Rewrite URLs to redirect

Suppose you currently have the following statement:

```
response.sendRedirect ("http://myhost/store/catalog");
```

Change the servlet to call the encodeRedirectURL method before sending the URL to the output stream:

```
response.sendRedirect (response.encodeRedirectURL ("http://myhost/store/catalog"));
```

The encodeURL method and encodeRedirectURL method are part of the HttpServletResponse object. These calls check to see if URL rewriting is configured before encoding the URL. If it is not configured, the calls return the original URL.

If both cookies and URL rewriting are enabled and the response.encodeURL method or encodeRedirectURL method is called, the URL is encoded, even if the browser making the HTTP request processed the session cookie.

You can also configure session support to enable protocol switch rewriting. When this option is enabled, the product encodes the URL with the session ID for switching between HTTP and HTTPS protocols.

Supply a servlet or JSP file as an entry point

The entry point to an application, such as the initial screen presented, may not require the use of sessions. However, if the application in general requires session support (meaning some part of it, such as a servlet, requires session support), then after a session is created, all URLs are encoded to perpetuate the session ID for the servlet (or other application component) requiring the session support.

The following example shows how you can embed Java code within a JSP file:

```
<%
response.encodeURL ("/store/catalog");
%>
```

Session tracking with SSL information (Deprecated)

Note: Session tracking using the SSL ID is deprecated in WebSphere Application Server version 7.0. You can configure session tracking to use cookies or URL rewriting.

No special programming is required to track sessions with Secure Sockets Layer (SSL) information.

To use SSL information, turn on **Enable SSL ID tracking** in the session management property page. Because the SSL session ID is negotiated between the Web browser and HTTP server, this ID cannot survive an HTTP server failure. However, the failure of an application server does not affect the SSL session ID if an external HTTP server is present between WebSphere Application Server and the browser.

SSL tracking is supported for the IBM HTTP Server and iPlanet Web servers only. You can control the lifetime of an SSL session ID by configuring options in the Web server. For example, in the IBM HTTP Server, set the configuration variable SSLV3TIMEOUT to provide an adequate lifetime for the SSL session ID. An interval that is too short can cause a premature termination of a session. Also, some Web browsers might have their own timers that affect the lifetime of the SSL session ID. These Web browsers may not leave the SSL session ID active long enough to serve as a useful mechanism for session tracking. The internal HTTP Server of WebSphere Application Server also supports SSL tracking.

When using the SSL session ID as the session tracking mechanism in a cloned environment, use either cookies or URL rewriting to maintain session affinity. The cookie or rewritten URL contains session affinity information that enables the Web server to properly route a session back to the same server for each request.

Session recovery support

For session recovery support, WebSphere Application Server provides distributed session support in the form of database sessions. You can use session recovery support when the user's session data must be maintained across a server restart or when the user's session data is too valuable to lose through an unexpected server failure.

All the attributes set in a session must implement java.io.Serializable if the session requires external storage. In general, consider making all objects held by a session serialized, even if immediate plans do not call for session recovery support. If the Web site grows, and session recovery support becomes necessary, the transition occurs transparently to the application if the sessions only hold serialized objects. If not, a switch to session recovery support requires coding changes to make the session contents serialized.

Session management tuning

WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

The following table summarizes the features, including whether they apply to sessions tracked in memory, in a database, with memory-to-memory replication, or all. Some features are easily manipulated using administrative settings; others require code or database changes.

Feature or option	**Goal**	**Applies to sessions in memory, database, or memory-to-memory**
Write frequency	Minimize database write operations.	Database
Session affinity	Access the session in the same application server instance.	All
Multirow schema	Fully utilize database capacities.	Database
Base in-memory session pool size	Fully utilize system capacity without overburdening system.	All
Write contents	Allow flexibility in determining what session data to write	Database
Scheduled invalidation	Minimize contention between session requests and invalidation of sessions by the Session Management facility. Minimize write operations to database for updates to last access time only.	Database
Tablespace and row size	Increase efficiency of write operations to database.	Database (DB2® only)

HTTP sessions: Resources for learning

Use the following links to find relevant supplemental information about HTTP sessions. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks® that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:

Programming model and decisions
- Improving session persistence performance with DB2

Programming instructions and examples
- Java Servlet documentation, tutorials, and examples site

Programming specifications
- Java Servlet 2.4 API specification download site
- J2EE 1.4 specification download site

Scheduled invalidation
Instead of relying on the periodic invalidation timer that runs on an interval based on the session timeout parameter, you can set specific times for the session management facility to scan for invalidated sessions in a distributed environment.

When used with distributed sessions, this feature has the following benefits:
- You can schedule the scan for invalidated sessions for times of low application server activity, avoiding contention between invalidation scans of database or another WebSphere Application Server instance and read and write operations to service HTTP session requests.
- Significantly fewer external write operations can occur when running with the End of Service Method Write mode because the last access time of the session does not need to be written out on each HTTP request. (Manual Update options and Time Based Write options already minimize the writing of the last access time.)

Usage considerations
- The session manager invalidates sessions only at the scheduled time, therefore sessions are available to an application if they are requested before the session is invalidated.
- With scheduled invalidation configured, HttpSession timeouts are not strictly enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- HttpSessionBindingListener processing is handled at the configured invalidation times unless the HttpSession.invalidate method is explicitly called.
- The HttpSession.invalidate method immediately invalidates the session from both the session cache and the external store.
- The periodic invalidation thread still runs with scheduled invalidation. If the current hour of the day does not match one of the configured hours, sessions that have exceeded the invalidation interval are removed from cache, but not from the external store. Another request for that session results in returning that session back into the cache.
- When the periodic invalidation thread runs during one of the configured hours, all sessions that have exceeded the invalidation interval are invalidated by removal from both the cache and the external store.
- The periodic invalidation thread can run more than once during an hour and does not necessarily run exactly at the top of the hour.
- If you specify the interval for the periodic invalidation thread using the HttpSessionReaperPollInterval custom property, do not specify a value of more than 3600 seconds (1 hour) to ensure that invalidation processing happens at least once during each hour.

Base in-memory session pool size
The base in-memory session pool size number depends on the session support configuration.
- With in-memory sessions, session access is optimized for up to this number of sessions.

General memory requirements for the hardware system, and the usage characteristics of the e-business site, determines the optimum value.

Note that increasing the base in-memory session pool size can necessitate increasing the heap sizes of the Java processes for the corresponding WebSphere Application Servers.

Overflow in non-distributed sessions

By default, the number of sessions maintained in memory is specified by base in-memory session pool size. If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set overflow to *true*.

Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one HTTP request to the next.

When overflow is disallowed, the Session Management facility still returns a session with the HttpServletRequest getSession(true) method when the memory limit is reached, and this is an invalid session that is not saved.

With the WebSphere Application Server extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, an isOverflow method returns *true* if the session is such an invalid session. An application can check this status and react accordingly.

HTTP session invalidation

HTTP sessions are invalidated by calling the invalidate method on the session object or by specifying a specific time interval using the MaxInactiveInterval property.

Sessions that are invalidated explicitly by application code are invalidated immediately. Sessions that are not invalidated by application code are invalidated by the session manager. Session invalidation occurs regardless of session persistence configuration.

A session is a candidate for invalidation if it has not been accessed for a period that is longer than the specified session timeout, specified by the MaxInactiveInterval value. The session manager has an invalidation process thread that runs every X seconds to invalidate sessions that are eligible for invalidation.

The session manager uses a formula to determine the value of X, specified by the ReaperInterval property. The value of X is calculated based on the MaxInactiveInterval value that is specified in the session manager.

For example, for a maximum inactive interval less than 15 minutes, the ReaperInterval value is approximately 60 to 90 seconds. For a maximum inactive interval greater than 15 minutes, the ReaperInterval value is approximately 300 to 360 seconds.

A session is invalidated when the MaxInactiveInterval is exceeded and the ReaperInterval passes. After a session is eligible for invalidation, the invalidation thread must run for the session to be invalidated. Therefore, a session might not be invalidated for the sum of the MaxInactiveInterval and ReaperInterval value in seconds.

A session that has exceeded the MaxInactiveInterval but is not yet removed by the invalidation thread is still available for use. If that session is requested then it is returned to the client.

You can specify whether the session is invalidated immediately or after a specified time interval. For immediate invalidation the application should call the invalidate method. To invalidate a session at a specific time, you can set the ReaperInterval Web container custom property in seconds to specify the frequency of the invalidation thread.

Write operations

You can manually control when modified session data is written out to the database or to another WebSphere Application Server instance by using the sync method in the com.ibm.websphere.servlet.session.IBMSession interface. The manual update, end of service servlet and the time based write frequency modes are available to tune write frequency of session data.

This interface extends the javax.servlet.http.HttpSession interface. By calling the sync method from the service method of a servlet, you send any changes in the session to the external location. When manual update is selected as the write frequency mode, session data changes are written to an external location only if the application calls the sync method. If the sync method is not called, session data changes are lost when a session object leaves the server cache. When end of service servlet or time based is the write frequency mode, the session data changes are written out whenever the sync method is called. If the sync method is not called, changes are written out at the end of service method or on a time interval basis based on the write frequency mode that is selected.

```
IBMSession iSession = (IBMSession) request.getSession();
iSession.setAttribute("name", "Bob");

//force write to external store
iSession.sync( )
```

If the database is down or is having difficulty connecting during an update to session values, the sync method always makes three attempts before it finally creates a BackedHashtable.getConnectionError error. For each connection attempt that fails, the BackedHashtable.StaleConnectionException is created and can be found in the sync method. If the database opens during any of these three attempts, the session data in the memory is then persisted and committed to the database.

However, if the database is still not up after the three attempts, then the session data in the memory is persisted only after the next check for session invalidation. Session invalidation is checked by a separate thread that is triggered every five minutes. The data in memory is consistent unless a request for session data is issued to the server between these events. For example, if the request for session data is issued within five minutes, then the previous persisted session data is sent.

Sessions are not transactional resources. Because the sync method is associated with a separate thread than the client, the exception that is created does not propagate to the client, which is running on the primary thread. Transactional integrity of data can be maintained through resources such as enterprise beans.

Related reference

“Session management tuning” on page 14
WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

Tuning parameter settings

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management** → **Distributed environment settings** → **Custom tuning parameters**.

Tuning level

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

Very high (optimize for performance)

Write frequency	Time based
Write interval	300 seconds
Write contents	Only updated attributes
Schedule sessions cleanup	true
First time of day default	0
Second time of day default	2

High

Write frequency	Time based
Write interval	300 seconds
Write contents	All session attributes
Schedule sessions cleanup	false

Medium

Write frequency	End of servlet service
Write contents	Only updated attributes
Schedule sessions cleanup	false

Low (optimize for failover)

Write frequency	End of servlet service
Write contents	All session attributes
Schedule sessions cleanup	false

Custom settings

Write frequency default	Time based
Write interval default	10 seconds
Write contents default	All session attributes
Schedule sessions cleanup default	false

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management** → **Distributed environment settings** → **Custom tuning parameters** → **Custom settings**.

Write frequency

Specifies when the session is written to the persistent store.

End of servlet service	A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.

Manual update
A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.

Time based
Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds

Write contents
Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Only updated attributes
Only updated attributes are written to the persistent store.

All session attribute
All attributes are written to the persistent store.

Schedule sessions cleanup
Specifies when to clean the invalid sessions from a database or another application server instance.

Specify distributed sessions cleanup schedule
Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions.

When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.

First Time of Day (0 - 23)
Indicates the first hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Second Time of Day (0 - 23)
Indicates the second hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Best practices for using HTTP sessions
This topic presents best practices for the implementation of HTTP sessions.

Note: Browse the following recommendations for implementing HTTP sessions.

- **Enable Security integration for securing HTTP sessions**

  HTTP sessions are identified by session IDs. A session ID is a pseudo-random number generated at the runtime. Session hijacking is a known attack HTTP sessions and can be prevented if all the requests going over the network are enforced to be over a secure connection (meaning, HTTPS). But not every configuration in a customer environment enforces this constraint because of the performance impact of SSL connections. Due to this relaxed mode, HTTP session is vulnerable to hijacking and because of this vulnerability, WebSphere Application Server has the option to tightly integrate HTTP sessions and WebSphere Application Server security. Enable security in WebSphere Application Server so that the sessions are protected in a manner that only users who created the sessions are allowed to access them.

- **Release HttpSession objects using `javax.servlet.http.HttpSession.invalidate()` when finished.**

  HttpSession objects live inside the Web container until:
  - The application explicitly and programmatically releases it using the javax.servlet.http.HttpSession.invalidate method; quite often, programmatic invalidation is part of an application logout function.
  - WebSphere Application Server destroys the allocated HttpSession when it expires (default = 1800 seconds or 30 minutes). The WebSphere Application Server can only maintain a certain number of HTTP sessions in memory based on session management settings. In case of distributed sessions, when maximum cache limit is reached in memory, the session management facility removes the least recently used (LRU) one from cache to make room for a session.

  .
- **Avoid trying to save and reuse the HttpSession object outside of each servlet or JSP file.**

  The HttpSession object is a function of the HttpRequest (you can get it only through the req.getSession method), and a copy of it is valid only for the life of the service method of the servlet or JSP file. You *cannot* cache the HttpSession object and refer to it outside the scope of a servlet or JSP file.
- **Implement the java.io.Serializable interface when developing new objects to be stored in the HTTP session.**

  Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Implementing the java.io.Serializable interface allows the object to properly serialize when using distributed sessions. Classes that do not implement this interface will not have their states serialized or deserialized. Therefore, if a class does not implement the Serializable interface, the JVM cannot persist its state into a database or into another JVM. All subtypes of a serializable class are serializable. An example of this follows:

  ```
  public class MyObject implements java.io.Serializable {...}
  ```

  Make sure all instance variable objects that are not marked transient are serializable. You cannot cache a non-serializable object.

  In compliance with the Java Servlet specification, the distributed servlet container must create an IllegalArgumentException for objects when the container cannot support the mechanism necessary for migration of the session storing them. An exception is created only when you have selected distributable.
- **The HTTPSession API does not dictate transactional behavior for sessions.**

  Distributed HTTPSession support does not guarantee transactional integrity of an attribute in a failover scenario or when session affinity is broken. Use transactional aware resources like enterprise Java beans to guarantee the transaction integrity required by your application.
- **Ensure the Java objects you add to a session are in the correct class path.**

  If you add Java objects to a session, place the class files for those objects in the correct class path (the application class path if utilizing sharing across Web modules in an enterprise application, or the Web module class path if using the Servlet 2.2-complaint session sharing) or in the directory containing other servlets used in WebSphere Application Server.

  Because the HttpSession object is shared among servlets that the user might access, consider adopting a site-wide naming convention to avoid conflicts.
- **Avoid storing large object graphs in the HttpSession object.**

  In most applications each servlet only requires a fraction of the total session data. However, by storing the data in the HttpSession object as one large object, an application forces WebSphere Application Server to process all of it each time.
- **Utilize Session Affinity to help achieve higher cache hits in the WebSphere Application Server.**

  WebSphere Application Server has functionality in the HTTP Server plug-in to help with session affinity. The plug-in reads the cookie data (or encoded URL) from the browser and helps direct the request to the appropriate application or clone based on the assigned session key. This functionality increases use of the in-memory cache and reduces hits to the database or another WebSphere Application Server instance
- **Maximize use of session affinity and avoid breaking affinity.**

Using session affinity properly can enhance the performance of the WebSphere Application Server. Session affinity in the WebSphere Application Server environment is a way to maximize the in-memory cache of session objects and reduce the amount of reads to the database or another WebSphere Application Server instance. Session affinity works by caching the session objects in the server instance of the application with which a user is interacting. If the application is deployed in multiple servers of a server group, the application can direct the user to any one of the servers. If the users starts on server1 and then comes in on server2 a little later, the server must write all of the session information to the external location so that the server instance in which server2 is running can read the database. You can avoid this database read using session affinity. With session affinity, the user starts on server1 for the first request; then for every successive request, the user is directed back to server1. Server1 has to look only at the cache to get the session information; server1 never has to make a call to the session database to get the information.

You can improve performance by not breaking session affinity. Some suggestions to help avoid breaking session affinity are:

- Combine all Web applications into a single application server instance, if possible, and use modeling or cloning to provide failover support.
- Create the session for the frame page, but do not create sessions for the pages within the frame when using multi-frame JSP files. (See discussion later in this topic.)

- **When using multi-framed pages, follow these guidelines:**
  - Create a session in only one frame or before accessing any frame sets. For example, assuming there is no session already associated with the browser and a user accesses a multi-framed JSP file, the browser issues concurrent requests for the JSP files. Because the requests are not part of any session, the JSP files end up creating multiple sessions and all of the cookies are sent back to the browser. The browser honors only the last cookie that arrives. Therefore, only the client can retrieve the session associated with the last cookie. Creating a session before accessing multi-framed pages that utilize JSP files is recommended.
  - By default, JSP files get a HTTPSession using `request.getSession(true)` method. So by default JSP files create a new session if none exists for the client. Each JSP page in the browser is requesting a new session, but only one session is used per browser instance. A developer can use `<% @ page session="false" %>` to turn off the automatic session creation from the JSP files that do not access the session. Then if the page needs access to the session information, the developer can use `<%HttpSession session = javax.servlet.http.HttpServletRequest.getSession(false); %>` to get the already existing session that was created by the original session creating JSP file. This action helps prevent breaking session affinity on the initial loading of the frame pages.
  - Update session data using only one frame. When using framesets, requests come into the HTTP server concurrently. Modifying session data within only one frame so that session changes are not overwritten by session changes in concurrent frameset is recommended.
  - Avoid using multi-framed JSP files where the frames point to different Web applications. This action results in losing the session created by another Web application because the JSESSIONID cookie from the first Web application gets overwritten by the JSESSIONID created by the second Web application.
- **Secure all of the pages (not just some) when applying security to servlets or JSP files that use sessions with security integration enabled, .**

  When it comes to security and sessions, it is all or nothing. It does not make sense to protect access to session state only part of the time. When security integration is enabled in the session management facility, all resources from which a session is created or accessed must be either secured or unsecured. You cannot mix secured and unsecured resources.

  The problem with securing only a couple of pages is that sessions created in secured pages are created under the identity of the authenticated user. Only the same user can access sessions in other secured pages. To protect these sessions from use by unauthorized users, you cannot access these sessions from an unsecured page. When a request from an unsecured page occurs, access is denied and an `UnauthorizedSessionRequestException` error is created. (`UnauthorizedSessionRequestException` is a runtime exception; it is logged for you.)
- **Use manual update and either the sync() method or time-based write in applications that read session data, and update infrequently.**

With END_OF_SERVICE as write frequency, when an application uses sessions and anytime data is read from or written to that session, the LastAccess time field updates. If database sessions are used, a new write to the database is produced. This activity is a performance hit that you can avoid using the Manual Update option and having the record written back to the database only when data values update, not on every read or write of the record.

To use manual update, turn it on in the session management service. (See the tables above for location information.) Additionally, the application code must use the `com.ibm.websphere.servlet.session.IBMSession` class instead of the generic HttpSession. Within the IBMSession object there is a sync method. This method tells the WebSphere Application Server to write the data in the session object to the database. This activity helps the developer to improve overall performance by having the session information persist only when necessary.

Note: An alternative to using the manual updates is to utilize the timed updates to persist data at different time intervals. This action provides similar results as the manual update scheme.

- Implement the following suggestions to achieve high performance:
  - If your applications do not change the session data frequently, use Manual Update and the sync function (or timed interval update) to efficiently persist session information.
  - Keep the amount of data stored in the session as small as possible. With the ease of using sessions to hold data, sometimes too much data is stored in the session objects. Determine a proper balance of data storage and performance to effectively use sessions.
  - If using database sessions, use a dedicated database for the session database. Avoid using the application database. This helps to avoid contention for JDBC connections and allows for better database performance.
  - Verify that you have the latest fix packs for the WebSphere Application Server.
- Utilize the following tools to help monitor session performance.
  - Run the com.ibm.servlet.personalization.sessiontracking.IBMTrackerDebug servlet. - To run this servlet, you must have the servlet invoker running in the Web application you want to run this from. Or, you can explicitly configure this servlet in the application you want to run.
  - Use the WebSphere Application Server Resource Analyzer which comes with WebSphere Application Server to monitor active sessions and statistics for the WebSphere Application Server environment.
  - Use database tracking tools such as ″Monitoring″ in DB2. (See the respective documentation for the database system used.)

HTTP session manager troubleshooting tips

This article provides troubleshooting tips for problems creating or using HTTP sessions with your Web application hosted by WebSphere Application Server.

Here are some steps to take:

- See HTTP session aren't getting created or are getting dropped to see if your specific problem is discussed.
- View the JVM logs for the application server which hosts the problem application:
  - first, look at messages written while each application is starting. They will be written between the following two messages:

    ```
    Starting application: application
    ......................
    Application started: application
    ```

  - Within this block, look for any errors or exceptions containing a package name of com.ibm.ws.webcontainer.httpsession. If none are found, this is an indication that the session manager started successfully.
  - Error ″**SRVE0054E: An error occurred while loading session context and Web application**″ indicates that SessionManager didn't start properly for a given application.
  - Look within the logs for any Session Manager related messages. These messages will be in the format SESNxxxxE and SESNxxxxW for errors and warnings, respectively, where xxxx is a number identifying the precise error. Look up the extended error definitions in the Session Manager message table.

- See the Best practices for using HTTP Sessions section in the *Developing and deploying applications* PDF books for more details.
- To dynamically view the number of sessions as a Web application is running, enable performance monitoring for HTTP sessions. This will give you an indication as to whether sessions are actually being created.
- To learn how to view the HTTP session counters as the application runs, read the Monitoring performance with Tivoli® Performance Viewer chapter of the *Administering applications and their environment* PDF book.
- Alternatively, a special servlet can be invoked that displays the current configuration and statistics related to session tracking. This servlet has all the counters that are in performance monitor tool and has some additional counters.
  - Servlet name: **com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - It can be invoked from any Web module which is enabled to serve by class name. For example, using default_app, **http://localhost:9080/servlet/com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug**.
  - If you are viewing the module via the serve-by-class-name feature, be aware that it may be viewable by anyone who can view the application. You may wish to map a specific, secured URL to the servlet instead and disable the serve-servlets-by-classname feature.
- Enable tracing for the HTTP Session Manager component:
  - Use the trace specification **com.ibm.ws.session.*=all=enabled**. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.
  - If you are using persistent sessions based on memory replication, also enable trace for **com.ibm.ws.drs.***.
- If you are using **database-based persistent sessions**, look for problems related to the **data source** the Session Manager relies on to keep session state information. For details on diagnosing database related problems see the Errors accessing a datasource or connection pool in the *Administering applications and their environment* PDF book

Error message `SRVE0079E Servlet host not found` after you define a port

Error message SRVE0079E can occur after you define the port in WebContainer > HTTP Transports for a server, indicating that you do not have the port defined in your virtual host definitions. To define the port,

1. On the administrative console, go to Environment > Virtual Hosts > default_host> Host Aliases> New
2. Define the new port on host ″*″

The application server gets EC3 - 04130007 ABENDs

To prevent an EC3 - 04130007 abend from occuring on the application server, change the HTTP Output timeout value. The custom property *ConnectionResponseTimeout* specifies the maximum number of seconds the HTTP port for an individual server can wait when trying to read or write data. For instructions on how to set *ConnectionResponseTimeout*, see HTTP transport custom properties section of the *Administering applications and their environment* PDF book.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you don’t find your problem listed there contact IBM support.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

HTTP session problems

This article provides troubleshooting information related to creating or using Hypertext Transfer Protocol (HTTP) sessions.

To view and update the session manager settings discussed here, use the administrative console. Select the application server that hosts the problem application, then under **Additional properties**, select **Web Container**, then **Session manager**.

What kind of problem are you having?

- HTTP Sessions are not getting created, or are lost between requests.
- Session is shared across multiple browsers on same client machine.
- Session is not getting invalidated immediately after specified session timeout interval.
- Unwanted sessions are being created by JavaServer Pages.
- Session data intended for one client is seen by another client.
- A ClassCastException error occurs during failover of a session that contains an Enterprise JavaBeans™ (EJB) reference.
- Users are not logged out after HTTP session timer expires

If your problem is not described here, or none of these steps fixes the problem:

- Review “HTTP session manager troubleshooting tips” on page 22 for general steps on debugging session-manager related problems.
- Review Task overview: Managing HTTP sessions in the *Administering applications and their environment* PDF book for information on how to configure the session manager, and best practices for using it.
- Check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).
- If you don’t find your problem listed there contact IBM support.

HTTP sessions are not getting created, or are lost between requests

By default, the session manager uses cookies to store the session ID on the client between requests. Unless you intend to avoid cookie-based session tracking, ensure that cookies are flowing between WebSphere Application Server and the browser:

- Make sure the **Enable cookies** check box is checked under the **Session tracking Mechanism** property.
- Make sure cookies are enabled on the browser you are testing from or from which your users are accessing the application.
- Check the Cookie domain specified on the SessionManager (to view the or update the cookie settings, in the **Session tracking mechanism->enable cookies** property, click **Modify**).
  - For example, if the cookie domain is set as ″.myCom.com″, resources should be accessed using that domain name. Example: http://www.myCom.com/myapp/servlet/sessionservlet.
  - If the domain property is set, make sure it begins with a dot (.). Certain versions of Netscape do not accept cookies if domain name doesn’t start with a dot. Internet Explorer honors the domain with or without a dot. For example, if the domain name is set to *mycom.com*, change it to *.mycom.com* so that both Netscape and Internet Explorer honor the cookie.

    Note: When the servers are on different hosts, ensure that session cookies flow to all the servers by configuring a front-end router such as a Web server with the plug-in or setting the Cookie domain.
- Check the **Cookie path** specified on the SessionManager. Check whether the problem URL is hierarchically below the Cookie path specified. If not correct the Cookie path.
- If the `Cookie maximum age` property is set, ensure that the client (browser) machine’s date and time is the same as the server’s, including the time zone. If the client and the server time difference is over the ″Cookie maximum age″ then every access would be a new session, since the cookie expires after the access.

- If you have multiple Web modules within an enterprise application that track sessions:
  - If you want to have different session settings among Web modules in an enterprise application, ensure that each Web module specifies a different cookie name or path, or
  - If Web modules within an enterprise application use a common cookie name and path, ensure that the HTTP session settings, such as Cookie maximum age, are the same for all Web modules. Otherwise cookie behavior is unpredictable, and depends upon which application creates the session. Note that this does not affect session data, which is maintained separately by Web module.
- Check the cookie flow between browser and server:
  1. On the browser, enable ″cookie prompt″. Hit the servlet and make sure cookie is being prompted.
  2. On the server, enable SessionManager trace. Enable tracing for the HTTP session manager component, by using the trace specification ″com.ibm.ws.session.*=all=enabled″. After trace is enabled, exercise your session-using servlet or JSP, then follow the instructions for dumping and browsing the trace output .
  3. Access the session servlet from the browser.
  4. The browser prompts for the cookie; note the jsessionid.
  5. Reload the servlet, note down the cookie if a new cookie is sent.
  6. Check the session trace and look for the session ID and trace the request by the thread. Verify that the session is stable across Web requests:
     - Look for **getIHttpsession(...)** which is start of session request.
     - Look for **releaseSession(..)** which is end of servlet request.
- If you are using URL rewriting instead of cookies:
  - Ensure there are no static HTML pages on your application's navigation path.
  - Ensure that your servlets and JSP files are implementing URL rewriting correctly. For details and an example see the Session tracking options section in the *Developing and deploying applications* PDF book.
-

  Note: Session tracking using the SSL ID is deprecated in WebSphere Application Server version 7.0. You can configure session tracking to use cookies or modify the application to use URL rewriting.

  If you are using SSL as your session tracking mechanism:
  - Ensure that you have SSL enabled on your IBM HTTP Server or iPlanet HTTP server.
  - Review the Session tracking options section in the *Developing and deploying applications* PDF book..

Session is shared across multiple browsers on same client machine

This behavior is browser-dependent. It varies between browser vendors, and also may change according to whether a browser is launched as a new process or as a subprocess of an existing browser session (for example by hitting Ctl-N on Windows®).

The Cookie maximum age property of the session manager also affects this behavior, if cookies are used as the session-tracking mechanism. If the maximum age is set to some positive value, all browser instances share the cookies, which are persisted to file on the client for the specified maximum age time.

Session is not getting invalidated immediately after specified session timeout interval

The SessionManager invalidation process thread runs every *x* seconds to invalidate any invalid sessions, where x is determined based on the session timeout interval specified in the session manager properties. For the default value of 30 minutes , *x* is around 300 seconds. In this case, it could take up to 5 minutes (300 seconds) beyond the timeout threshold of 30 minutes for a particular session to become invalidated.

Unwanted sessions are being created by JavaServer Pages

As required by the JavaServer Pages (JSP) specification, JSP pages by default perform a request.getSession(true), so that a session is created if none exists for the client. To prevent JSP pages from creating a new session, set the session scope to **false** in the .jsp file using the page directive as follows:

```
<% @page session="false" %>
```

Session data intended for one client is seen by another client

In rare situations, usually due to application errors, session data intended for one client might be seen by another client. This situation is referred to as session data crossover. When the *DebugSessionCrossover* custom property is set to true, code is enabled to detect and log instances of session data crossover. Checks are performed to verify that only the session associated with the request is accessed or referenced. Messages are logged if any discrepancies are detected. These messages provide a starting point for debugging this problem. This additional checking is only performed when running on the WebSphere-managed dispatch thread, not on any user-created threads.

For additional information on how to set this property, see article, Web container custom properties in the *Administering applications and their environment* PDF book.

Users are not logged out after the HTTP session timer expires

If users of WebSphere Application Server log onto an application and sit idle longer than the specified HTTP session timeout value, the user information is not invalidated and user credentials stay active until LTPA token timeout occurs.

After you apply PK25740, complete the following steps to log out users from the application after the HTTP session has expired.

1. In the administrative console, click **Security > Global security**.
2. Under Custom properties, click **New**.
3. In the Name field, enter com.ibm.ws.security.web.logoutOnHTTPSessionExpire.
4. In the Values field, enter true.
5. Click Apply and Save to save the changes to your configuration.
6. Resynchronize and restart the server.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/allproducts/index.html

Modifying the default Web container configuration

About this task

A Web container handles requests for servlets, JavaServer Pages (JSP) files, and other types of files that include server-side code. The Web container creates servlet instances, loads and unloads servlets, creates and manages request and response objects, and performs other servlet management tasks. The Web server plug-ins, provided by the product, help supported Web servers to pass servlet requests to Web containers.

If the property to start servlets during application server startup is enabled, part of its startup process calls the Servlet.init method on its servlets when you start the Web container. Therefore, when the Web container starts and calls the init method, other components such as Naming and Work Load Management

might not be fully started yet. As a result, application server related calls may not work because all of the application server components might not be ready yet. Once the application server is 'ready for e-business', it is completely ready. If application server related calls fail during Servlet.init method, you can either:

- Start the servlet manually when the server is ready for e-business instead of starting the servlet upon startup or
- You can choose not to make application server related calls in the servlet's init method.

The Web container is created initially with default properties values suitable for simple Web applications. However, these values might not be appropriate for more complex Web applications.

Your application is considered complex if it requires any of the following features:
- Additional virtual host aliases
- Servlet caching
- Persistent HTTP session support
- Session tracking support with URL rewriting
- Special Web container transport chain settings
- Asynchronous or remote dispatching
- No request or response pooling

Make the following configuration changes if you have a complex application:

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***. Then under Web container settings, click on one of the following:
   a. **Web container**, if your Web application requires a virtual host, other than the default_host, or requires servlet caching.
   b. **Web container transport chains**, if you need to reconfigure your HTTP connections.
2. If your application handles special client request loads, in the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***. Then under Additional Properties, click **Thread pools** to modify your thread pool settings.
3. If your application requires global settings for internal servlets for Web archive (WAR) files packaged by third-party tools, in the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container**. Then under Additional Properties, click **Custom properties** and enter the appropriate custom property.

Web container settings

Use this page to configure the Web container settings.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container**.

Default virtual host

Specifies a virtual host that enables a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

Select a virtual host option:

default_host

The product provides a default virtual host with some common aliases such as the machine IP address, short host name, and fully qualified host name. The alias comprises the first part of the path for accessing a resource such as a servlet. For example, it is `localhost:9080` in the request `http://localhost:9080/myServlet`.

admin_host

This is another name for the application server; also known as *server1* in the base installation. This process supports the use of the administrative console.

proxy_host
The virtual host called proxy_host, includes default port definitions, port 80 and 443, which are typically initialized as part of the proxy server initialization. Use this proxy host as appropriate with routing rules associated with the proxy server.

Enable servlet caching
Specifies that if a servlet is invoked once and it generates output to be cached, a cache entry is created containing not only the output, but also side effects of the invocation. These side effects can include calls to other servlets or JavaServer Pages (JSP) files, as well as metadata about the entry, including timeout and entry priority information.

Portlet fragment caching requires that servlet caching is enabled. Therefore, enabling portlet fragment caching automatically enables servlet caching. Disabling servlet caching automatically disables portlet fragment caching.

Disable servlet request and response pooling
Specifies to disable the pooling of servlet request and servlet response objects that are pooled by the Web container. When you disable pooling of servlet request and servlet response objects, new servlet request and servlet response objects are created for each request.

When you disable pooling of servlet request and servlet response objects, new servlet request and servlet response objects are created for each request, which can negatively affect performance, but provides protection from any unforeseen pooling bugs.

Web container custom properties
You can configure name-value pairs of data, where the name is a property key and the value is a string value that you can use to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console. The following is a list of some of the available Web container custom properties.

To specify Web container custom properties:
1. In the administrative console click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container**.
2. Under **Additional Properties** select **Custom Properties**.
3. On the Custom Properties page, click **New**.
4. On the settings page, enter the name of the custom property that you want to configure in the **Name** field and the value that you want to set it to in the **Value** field.
5. Click **Apply** or **OK**.
6. Click **Save** on the console task bar to save your configuration changes.
7. Restart the server.

The following is a list of custom properties provided with the Application Server. “JavaServer Pages specific Web container custom properties” on page 75 and HTTP transport custom properties are listed in a separate topic.

com.ibm.ws.webcontainer.disallowAllFileServing
This property is for disabling file serving on all applications on a specific application server. You can enable file serving on a global level across a given application server by using the fileServingEnabled custom property. However, the fileServingEnabled property is overridden by the specific deployment information of each application. Therefore, the current fileServingEnabled custom property only applies as a backup in case an application does not define the fileServingEnabled setting itself. To globally override this setting on a specific application server to prevent the application server from serving static files regardless of their individual deployment settings, set the com.ibm.ws.webcontainer.disallowAllFileServing Web container custom property to `true` using the following name-value pair.

Name	Value
com.ibm.ws.webcontainer.disallowAllFileServing	true

Disallow the use of the serving servlets by class name

When the serveServletsByClassnameEnabled property is enabled, it is possible to access servlets directly, resulting in a possible security exposure. Define the following custom property to disallow the use of the serveServletsByClassnameEnabled property across the entire application server level.

Name	Value
com.ibm.ws.webcontainer.disallowserveservletsbyclassname	true

Defining classes that cannot be served by class name

The com.ibm.ws.webcontainer.donotservebyclassname custom property specifies a list of classes that cannot be served by the class name.

Name	Value
com.ibm.ws.webcontainer.donotservebyclassname	A semi-colon delimited list of classes to be completely disallowed from being served by class name.

Writing chunks of data using synchronous writes

By default, the Web container uses asynchronous writes to write response data in chunks up to the response buffer size. For larger responses that are greater than the response buffer size, the Web container continues to buffer response data into memory while waiting for an asynchronous write of a response data chunk to complete. This can result in part of a large response held in memory, which can lead to high memory usage and potentially an out of memory error. An application server hang may also occur when a server is simultaneously processing more requests than Web container defined threads.

If the com.ibm.ws.webcontainer.channelwritetype property is set to `sync`, synchronous writing is used, otherwise asynchronous writing is used by default. With synchronous writing, response data are written synchronously in chunks of up to the value of responsebuffersize and no response data are buffered into memory while waiting for a synchronous write of a response data chunk to complete. As a result, the approximate maximum amount of response data that is held in memory is equal to the responsebuffersize multiplied by the number of Web container threads. The maximum number of requests that can be processed simultaneously by the Web container is limited by the number of Web container threads. Additional requests are queued, waiting for a request that is in process to complete.

The responsebuffersize Web container custom property defines the maximum amount of response data written by the Web container in a single chunk, and is 32k by default. As a result, it is used to change the number of writes needed by the Web container to send complete response data. However, if an application flushes response data, any response data held by the Web container is immediately written irrespective of the responsebuffersize.

Use the following name-value pair to write chunks of data using synchronous writes.

Name	Value
com.ibm.ws.webcontainer.channelwritetype	sync

Recognizing filter mappings to ″*″ as a mapping to ″/*″

When processing a request, the Web container recognizes servlet mappings to ″*″ as the same as servlet mappings to ″/*″. To provide the same behavior with filter mapping, set the com.ibm.ws.webcontainer.mapFiltersToAsterisk custom property to `true`. Setting the com.ibm.ws.webcontainer.mapFiltersToAsterisk custom property to `true` causes the Web container to recognize filter mappings to ″*″ as a filter mapping to ″/*″ . This custom property is not case sensitive.

Name	Value
com.ibm.ws.webcontainer.mapFiltersToAsterisk	true

Suppressing the HTML output of error text

During a recursive error that an application specified error page cannot handle, the stack trace and error message are outputted as an HTML page. This information includes class names and program information that the application developer does not want expose to the user.

You can set the com.ibm.ws.webcontainer.suppressHtmlRecursiveErrorOutput Web container custom property to suppress the HTML output of the error text, without changing the internal logging of the message. Set the custom property com.ibm.ws. webcontainer.suppressHtmlRecursiveErrorOutput to `true` to disable the HTML output of the error message to the user and present the user with blank page with a 500 error code.

Name	Value
com.ibm.ws.webcontainer.suppressHtmlRecursiveErrorOutput	true

DebugSessionCrossover

The *DebugSessionCrossover* custom property enables code to perform additional checks to verify that only the session associated with the request is accessed or referenced. Messages are logged if any discrepancies are detected.

Note: The use of the *DebugSessionCrossover* property as a Web container custom property is deprecated. You can now define it as a session management custom property.

To enable session data crossover detection, use the following name-value pair:

Name	Value
DebugSessionCrossover	true

See article, “HTTP session problems” on page 24, for additional information.

DecodeUrlAsUTF8

The UTF-8 encoded URL feature, which provides UTF-8 encoded Uniform Resource Locators (URLs) to support the double-byte characters in URLs is enabled by default. You can prevent the Web container from explicitly decoding URLs in UTF-8 and have them use the ISO-8859 standard as per the current HTTP specification by using the following name-value pair.

Name	Value
DecodeUrlAsUTF8	false

enableInProcessConnections

Use the enableInProcessConnections custom property to reduce response times and to reduce the number of threads that are used to service a request, which reduces the potential for a deadlock. There is an optimized communication path between a Web services client application and a Web container that are located in the same application server process. Requests from the Web services client that are normally sent to the Web container using a network connection are delivered directly to the Web container using an optimized local path. The local path is available because the Web services client application and the Web container are running in the same process. This optimized communication path is disabled by default. Before enabling this property, make sure that wild cards are not specified for the Web container ports. Use specific ports for the Web container when the optimized communication path is enabled. To enable the optimized communication path, use the following name-value pair:

Name	Value
enableInProcessConnections	true

See article, Web services client to Web container optimized communication, for additional information.

fileServingEnabled

fileServingEnabled, directoryBrowsingEnabled, and similar properties are global settings for internal servlets. Web application archive (WAR) files that are packaged using third-party tools cannot specify behavior for the services that are exposed by the Web container internal servlets. You can globally enable and disable internal servlets for all Web applications at the Web container level by creating name-value pairs.

Name	Value
fileServingEnabled	true
directoryBrowsingEnabled	true
serveServletsByClassnameEnabled	true

Settings that are defined in an assembly tool take precedence over the global settings that are set through the custom properties at the Web container level.

Web application deployment extensions continue to hold configuration information for the services that are provided by the internal servlets, and take precedence over the global settings that are set through the custom properties at the Web container level.

Invalidating sessions after specified time has elapsed

The *ForceSessionInvalidationMultiple* custom property indicates whether the session manager should wait indefinitely for a request to complete before attempting to invalidate the session, or attempt to invalidate a session after the specified time limit has elapsed. The default value for this property is 1.

- If you specify 0 (zero) for this custom property, the session manager waits indefinitely until a request is complete before attempting to invalidate the session.

  If your requests normally are not bound by a response time limit, specify 0 for this property.
- If you specify a positive integer, such as 1, 2, or 3 for this custom property, even if a session is not known to have completed, the session manager attempts to invalidate the session if the indicated time period since the last access occurred has elapsed. This time period is the result of multiplying the value specified for this property and the value specified for the Session Timeout property. For example, if you specify 2 minutes for the Session Timeout property and 2 for the ForceSessionInvalidationMultiple property, the session manager attempts to invalidate the session after 4 minutes.

  If you want to invalidate your sessions after a certain amount of time has elapsed, specify the appropriate positive integer for this property.

Name	Value
ForceSessionInvalidationMultiple	1

httpsIndicatorHeader

The custom property *httpsIndicatorHeader* manages HTTPS requests that are forwarded to an application server from an SSL offloader that is used in front of WebSphere Application Server. When an HTTPS request is received by a SSL offloader it is redirected over HTTP to an application server using WebSphere Application Server. The SSL offloader adds a header indicating the original request was over HTTP. The *httpsIndicatorHeader* property specifies the header name added by the SSL box. The application server checks this indicator to determine if SSL is required. If it determines the request is SSL over HTTP, an HTTPS scheme is chosen.

Name	Value
httpsIndicatorHeader	*Request header key name*

HttpSessionIdReuse

The custom property *HttpSessionIdReuse* determines whether the session manager can use the session ID sent from a browser to preserve session data across Web applications that are running in an environment that is not configured for session persistence. In a multi-JVM environment that is not configured for session persistence setting this property to `true` enables the session manager to use the same session information for all of a user's requests even if the Web applications that are handling these requests are governed by different JVMs. The default value for this property is `false`.

Note: The use of the *HttpSessionIdReuse* property as a Web container custom property is deprecated. You can now define it as a session management custom property.

To enable the session manager ton use the session ID sent from a browser to preserve session data across Web applications that are running in an environment that is not configured for session persistence, use the following name-value pair:

Name	Value
HttpSessionIdReuse	true

Global configuration of servlet listeners

The servlet specification supports applications registering listeners for servlet-related events on an individual application basis through the `web.xml` descriptor. However, using the listeners custom property, a server can listen to servlet events across Web applications. To implement global listening, a listener is registered at the Web container level and is propagated to all of the installed and new Web applications. This global behavior of internal servlet listeners is controlled by the listeners custom property by using the following name-value pair format.

Name	Value
listeners	*listener_class*

The values for this property is a string specifying a comma separated list of listener classes. The listener supplied must implement standard listener classes from the Java Servlet API or IBM listener extension classes.

Using Uniform Resource Locators (URLs) without leading front slashes (/)

WebSphere Application Server 5.x supports Uniform Resource Locators (URLs) without leading front slashes (/) and to preserve compatibility, You can set the custom property, prependSlashToResource to `true`. To ignore the specification and consider URLs without the leading front slash, use the following name-value pair.

Name	Value
prependSlashToResource	true

UseOracleBLOB

The *UseOracleBLOB* custom property creates the HTTP session database table using the Binary Large Object (BLOB) data type for the medium column. This property increases performance of persistent sessions when Oracle databases are used. Due to an Oracle restriction, BLOB support requires use of the oci database driver for Oracle for more than 4000 bytes of data. You must also ensure that a new sessions table is created before the server is restarted by dropping your old sessions table or by changing the datasource definition to reference a database that does not contain a sessions table.

Note: The use of the *UseOracleBLOB* property as a Web container custom property is deprecated. You can now define it as a session management custom property.

To create a sessions table using the BLOB data type, use the following name-value pair:

Name	Value
UseOracleBLOB	true

Convert start and end attributes of the repeat tag to strings

Set the com.ibm.wsspi.jsp.convertAttrValueToString Web container custom property to `true` to convert start and end attributes of the repeat tag to strings before they are used.

Name	Value
com.ibm.wsspi.jsp.convertAttrValueToString	true

Disable the commons-el expression cache

Set the com.ibm.wsspi.jsp.disableElCache Web container custom property to `true` to disable the commons-el expression cache if you are experiencing out of memory conditions because the hash maps are held by the expression evaluator.

Name	Value
com.ibm.wsspi.jsp.disableElCache	true

Create a FileNotFoundException exception when a JSP file is not found

Set the com.ibm.ws.webcontainer.throwMissingJspException custom property to `true` to create a FileNotFoundException when a resource that is included by a JSP file is missing. If this property is not set to `true`, an error page is displayed.

Name	Value
com.ibm.ws.webcontainer.throwMissingJspException	true

com.ibm.ws.webcontainer.suppressServletExceptionLogging

If a servlet creates an exception, it is logged in the systemErr log file. For some applications, this is undesirable as it may fill up the log. In WebSphere Application Server Version 5, servlet exceptions are created but they are not logged in the systemErr log file. Beginning with WebSphere Application Server Version 6.0.2, all servlet exceptions are logged. You can use the following custom property to specify that the Web container does not log the servlet exceptions that are created in the systemErr file:

Name	Value
com.ibm.ws.webcontainer.suppressServletExceptionLogging	true

Invoke global session listeners

If a system application is the first to start, and the application attempts to load a global listener in a shared library that is associated with the server classloader, the application does not load that listener and prevents the listener from being loaded or invoked by a later non-system application. Set the com.ibm.ws.webcontainer.disableSystemAppGlobalListenerLoading custom property to true to prevent system applications from loading global listeners. When this property is set to `true`, the system application does not attempt to load the global listeners and later non-system applications can load them from a shared library associated with a server class loader.

Name	Value
com.ibm.ws.webcontainer.disableSystemAppGlobalListenerLoading	true

Controlling active request completion time
By default, when an application is stopped the Web container waits up to 60 seconds for each active request for a resource of that application to complete. You can now define the com.ibm.ws.webcontainer.ServletDestroyWaitTime Web container custom property to control the amount of time that the Web container waits for an active request to complete when the owning application is stopped.

Set the com.ibm.ws.webcontainer.ServletDestroyWaitTime custom property to an integer value, which specifies the number of seconds to wait for a request to complete. The default value is 60 seconds.

Name	Value
com.ibm.ws.webcontainer.ServletDestroyWaitTime	integer

Web module deployment settings
Use this page to configure an instance of Web module deployment.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Manage Modules** → ***Web_module_instance***.

URI
Specifies the relative location of the module within the application enterprise archive (EAR) file.

Alternate deployment descriptor
Specifies the alternate deployment descriptor for the module as defined in the application deployment descriptor according to the Java Platform, Enterprise Edition (Java EE) specification.

Starting weight
Specifies the order that modules are started. Lower weighted modules are started before higher weighted modules.

Class loader order
Specifies whether the class loader searches in the parent class loader or in the application class loader first to load a class. The standard for development kit class loaders and product class loaders is `Classes loaded with parent class loader first`. By specifying `Classes loaded with application class loader first`, your application can override classes contained in the parent class loader, but this action can potentially result in ClassCastException or LinkageErrors if you have mixed use of overridden classes and non-overridden classes.

The options are `Classes loaded with parent class loader first` and `Classes loaded with local class loader first (parent last)` . The default is to search in the parent class loader before searching in the application class loader to load a class.

Data type String
Default Classes loaded with parent class loader first

Context root for Web modules settings
Use this page to specify the context root for Web modules during or after installation of an application onto a WebSphere Application Server deployment target.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Context root for Web modules**. This panel is the same as the Context root for Web modules panel on the application installation and update wizards.

Web module
Specifies the name of a Web module in the application that you are installing or that you are viewing after installation.

URI
Specifies the location of the module relative to the root of the application EAR file.

Context root
Specifies the context root of the Web application (WAR).

A context root for each Web module is defined in the application deployment descriptor during application assembly. Use this field to assign a different context root to a Web module. The context root is combined with the defined servlet mapping (from the WAR file) to compose the full URL that users type to access the servlet. For example, if the context root is `/gettingstarted` and the servlet mapping is `MySession`, then the URL is `http://host:port/gettingstarted/MySession`.

Environment entries for Web modules settings
Use this page to configure the environment entries of Web modules such as servlets and JavaServer Pages (JSP) files.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Environment entries for Web modules**. This page is the same as the Environment entries for Web modules panel on the application installation and update wizards.

Module
Specifies the name of a module in the `web.xml` Web application.

URI
Specifies the location of the module relative to the root of the application (EAR file).

Name
Specifies the name of the environment entry that you are editing or viewing. The environment entry is the `Env-entry` property in the module `web.xml` file.

Type
Specifies a data type for the environment entry defined by the `Env-entry` property in the module `web.xml` file.

Description
Specifies information on the environment entry.

Value
Specifies an editable value for the environment entry defined by the `Env-entry` property in the module `web.xml` file.

Web container troubleshooting tips

If you are having problems starting a Web module, or accessing resources within a particular Web module:
- View the JVM logs and process logs for the application server which hosts the problem Web modules, and look for messages in the JVM output file which indicate that the web module has started successfully. You should see messages similar to the following:

```
WebContainer  A SRVE0161I: IBM WebSphere Application Server - Web Container.
Copyright IBM Corp. 1998-2002
WebContainer  A SRVE0169I: Loading Web Module: [module_name]
```

```
ApplicationMg A WSVR0221I: Application started: [application_name]
HttpTransport A SRVE0171I: Transport http is listening on port [port_number]
[server_name] open for e-business in profile_root/logs/[server_name]/SystemOut.log
```

- For specific problems that can cause servlets, HTML files, and JavaServer Pages (JSP) files not to be served, see Web resource (JSP file, servlet, HTML file, image) does not display .
- For a detailed trace of the run-time behavior of the Web container, enable trace for the component com.ibm.ws.webcontainer using `com.ibm.ws.webcontainer*=all`.

If application server related calls fail during Servlet.init method, you can either:

- Initialize the servlet manually by making a single request to that servlet in your browser when the server is ready for e-business instead of starting the servlet upon startup or
- You can choose not to make application server related calls in the servlet's init method.

If the property to start servlets during application server startup is enabled, part of its startup process calls the Servlet.init method on its servlets when you start the Web container. Therefore, when the Web container is starts and calls the init method, other components such as Naming and Work Load Management may not be fully started yet. As a result, application server related calls may not work since all of the application server components may not be ready yet. Once the application server is 'ready for e-business', it is completely ready.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, tech notes, and fixes). If you don't find your problem listed there contact IBM support.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Disabling servlet pooling: Best practices and considerations

This topic provides usage examples of when you may want to disable servlet pooling. You may want to disable request and response pooling if your application is creating threads inside of the application or if you are concerned about the Web container reusing request and response objects.

Disabling request and response pooling

- Application is creating threads inside of the application.

  The Servlet 2.4 specification states the following:

  > **SRV.4.10 Lifetime of the Request Object** Each request object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method. Containers commonly recycle request objects in order to avoid the performance overhead of request object creation. The developer must be aware that maintaining references to request objects outside the scope described above is not recommended as it may have indeterminate results.
  >
  > **SRV.5.6 Lifetime of the Response Object** Each response object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method. Containers commonly recycle response objects in order to avoid the performance overhead of response object creation. The developer must be aware that maintaining references to response objects outside the scope described above may lead to non-deterministic behavior.

- If you are concerned about the Web container reuse of reusing request and response objects. Since these objects are reused, there is the potential for two requests in two separate applications to have access to the same request or response object as described in the Thread Safety section of Servlet 2.4.

SRV.2.3.3.3 Thread Safety Implementations of the request and response objects are not guaranteed to be thread safe. This means that they should only be used within the scope of the request handling thread.

References to the request and response objects should not be given to objects executing in other threads as the resulting behavior may be nondeterministic. If the thread created by the application uses the container-managed objects, such as the request or response object, those objects must be accessed only within the servlet's service life cycle and such thread itself should have a life cycle within the life cycle of the servlet's service method because accessing those objects after the service method ends may cause undeterministic problems. Be aware that the request and response objects are not thread safe. If those objects were accessed in the multiple threads, the access should be synchronized or be done through the wrapper to add the thread safety, for instance, synchronizing the call of the methods to access the request attribute, or using a local output stream for the response object within a thread.

It is important to note that disabling pooling prevents the Web container from recycling the servlet request and servlet response objects for subsequent requests. This creates additional overhead as a result of an increase in request and response object creation and the subsequent garbage collection of these discarded objects.

Task overview: Developing and deploying Web applications

About this task

A developer creates the files comprising a Web application, and then assembles the Web application components into a Web module. Next, the deployer (typically the developer in a unit-testing environment or the administrator in a production environment) installs the Web application on the server.

1. **(Optional)** Migrate existing Web applications to run in the new version of WebSphere Application Server.
2. Design the Web application and develop its code artifacts: Servlets, JavaServer Pages (JSP) files, and static files, as for example, images and Hyper Text Markup Language (HTML) files. See the "Web applications: Resources for learning" on page 80 topic for links to design documentation.

   JavaServer Pages programming tips:
   - Disable session state of JavaServer Pages files using `<%@ page language="java" contentType="text/html" session="false" %>` instead of `<%@ page language="java" contentType="text/html" %>`
   - Replace setProperties calls in your JavaServer Pages files with direct calls to the appropriate setxxx methods.
3. Develop the Web application, using WebSphere Application Server extensions to enhance its functionality.
4. Assemble the Web application into a Web module using an assembly tool. Web module assembly properties might include the ability to:
   - Configure servlet page lists.
   - Configure servlet filters.
   - Serve servlets by class name.

     Serving servlets by name or class name is triggered by setting the serveServletsbyclassnameEnabled property within IBM extensions. Use the invoker.patterns attribute to specify the patterns that trigger invocation of the server component and allows the serving of servlets by name or by class name. This value is a list separated by either a space, colon, or semicolon.
   - Enable file serving.

     In file serving, Web applications can serve static file types, such as HTML. File-serving attributes are used by the servlet that implements file-serving behavior.

5. Deploy the Web module or application module that contains the Web application.

   Following deployment, you might find it handy to use the tool that enables batch compiling of the JSP files for quicker initial response times.
6. **(Optional)** Troubleshoot your Web application.
7. **(Optional)** Modify the default Web container configuration in the application server in which you deployed the Web module or application module containing the Web application.
8. **(Optional)** Manage the deployed Web application.

Web applications

A Web application is comprised of one or more related servlets, JavaServer Pages technology (JSP files), and Hyper Text Markup Language (HTML) files that you can manage as a unit.

The files in a Web application are related in that they work together to perform a business logic function. The Web application is a concept supported by the Java Servlet Specification. Web applications are typically packaged as `.war` files.

Web modules

A Web module represents a Web application. A Web module is created by assembling servlets, JavaServer Pages (JSP) files, and static content such as Hypertext Markup Language (HTML) pages into a single deployable unit. Web modules are stored in Web archive (WAR) files, which are standard Java archive files.

A Web module contains:
- One or more servlets, JSP files, and HTML files.
- A deployment descriptor, stored in an Extensible Markup Language (XML) file.

  The file, named `web.xml`, declares the contents of the module. It contains information about the structure and external dependencies of Web components in the module and describes how the components are used at run time.

You can create Web modules as stand-alone applications, or you can combine Web modules with other modules to create Java Platform, Enterprise Edition (Java EE) applications. You can also install and run a Web module in the Web container of an application server.

web.xml file

The web.xml file provides configuration and deployment information for the Web components that comprise a Web application. Examples of Web components are servlet parameters, servlet and JavaServer Pages (JSP) definitions, and Uniform Resource Locators (URL) mappings.

The Java Servlet specification defines the web.xml deployment descriptor file in terms of an XML schema document. For backwards compatibility of applications written to the Java Servlet 2.2 Specification, Web containers are also required to support the Java Servlet 2.2 specification. For backwards compatibility of applications written to the Java Servlet 2.3 specification, Web containers are also required to support the Java Servlet 2.3 specification.

If you use Rational® Application Developer version 6 to create your portlets, you must remove the following reference to the std-portlet.tld from the web.xml file:

```
<taglib id="PortletTLD">
   <taglib-uri>http://java.sun.com/portlet</taglib-uri>
   <taglib-location>/WEB-INF/tld/std-portlet.tld</taglib-location>
  </taglib>
```

Location

The web.xml file must reside in the `WEB-INF` directory under the context of the hierarchy of directories that exist for a Web application.

For example, if the application is client.war, then the web.xml file is placed in the *profile_root*/`installedApps/cellName/client.ear/client.war/WEB-INF` directory (in a default installation), where the edition is either Base or Network Deployment, depending on which edition you are using.

Usage notes

- Is this file read-only?

  No
- Is this file updated by a product component?

  This file is updated by the assembly tool.
- If so, what triggers its update?

  The assembly tool updates the web.xml file when you assemble Web components into a Web module, or when you modify the properties of the Web components or the Web module.
- How and when are the contents of this file used?

  WebSphere Application Server functions use information in this file during the configuration and deployment phases of Web application development.

Sample file entry

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>Servlet 2.4 application</display-name>
 <filter>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <filter-class>tests.Filter.DoFilter_Filter</filter-class>
  <init-param>
   <param-name>attribute</param-name>
   <param-value>tests.Filter.DoFilter_Filter.SERVLET_MAPPED</param-value>
   </init-param>
 </filter>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>/DoFilterTest</url-pattern>
  <dispatcher>REQUEST</dispatcher>
 </filter-mapping>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>/IncludedServlet</url-pattern>
  <dispatcher>INCLUDE</dispatcher>
 </filter-mapping>
 <filter-mapping>
  <filter-name>ServletMappedDoFilter_Filter</filter-name>
  <url-pattern>ForwardedServlet</url-pattern>
  <dispatcher>FORWARD</dispatcher>
 </filter-mapping>
 <listener>
  <listener-class>tests.ContextListener</listener-class>
 </listener>
 <listener>
  <listener-class>tests.ServletRequestListener.RequestListener</listener-class>
 </listener>
 <servlet>
  <servlet-name>welcome</servlet-name>
  <servlet-class>WelcomeServlet</servlet-class>
 </servlet>
 <servlet>
```

```
 <servlet-name>ServletErrorPage</servlet-name>
 <servlet-class>tests.Error.ServletErrorPage</servlet-class>
</servlet>
<servlet>
 <servlet-name>IncludedServlet</servlet-name>
 <servlet-class>tests.Filter.IncludedServlet</servlet-class>
</servlet>
<servlet>
 <servlet-name>ForwardedServlet</servlet-name>
 <servlet-class>tests.Filter.ForwardedServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>welcome</servlet-name>
 <url-pattern>/hello.welcome</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>ServletErrorPage</servlet-name>
 <url-pattern>/ServletErrorPage</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>IncludedServlet</servlet-name>
 <url-pattern>/IncludedServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
 <servlet-name>ForwardedServlet</servlet-name>
 <url-pattern>/ForwardedServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
 <welcome-file>hello.welcome</welcome-file>
</welcome-file-list>
<error-page>
 <exception-type>java.lang.ArrayIndexOutOfBoundsException</exception-type>
 <location>/ServletErrorPage</location>
</error-page>
</web-app>
```

Default Application

WebSphere Application Server provides a default configuration that administrators can use to easily verify that the Application Server is running. When the product is installed, it includes an application server called *server1* and an enterprise application called *Default Application*.

Default Application contains a Web module called *DefaultWebApplication* and an enterprise bean Java archive (JAR) file called *Increment*. The *Default Application* provides a number of servlets, described below. These servlets are available in the product.

For additional code examples, visit the Samples Gallery. Learn how to locate and install the Samples Gallery by viewing the Samples Gallery reference page.

Snoop servlet

Use the Snoop servlet to retrieve information about a servlet request. This servlet returns the following information:

- Servlet initialization parameters
- Servlet context initialization parameters
- URL invocation request parameters
- Preferred client locale
- Context path
- User principal
- Request headers and their values
- Request parameter names and their values
- HTTPS protocol information

- Servlet request attributes and their values
- HTTP session information
- Session attributes and their values

The Snoop servlet includes security configuration so that when WebSphere Security is enabled, clients must supply a user ID and password to initiate the servlet.

The URL for the Snoop servlet is: `http://your.server.name:9080/HitCount.jsp`.

HelloHTML servlet

Use the HelloHTML pervasive servlet to exercise the PageList support provided by the WebSphere Web container. This servlet extends the PageListServlet, which provides APIs that allow servlets to call other Web resources by name or, when using the *Client Type detection* support, by type.

You can invoke the Hello servlet from an HTML browser, speech client, or most Wireless Application Protocol (WAP) enabled browsers using the URL: `http://your.server.name:9080/HitCount.jsp`.

Note: The PageList Servlet custom extension is deprecated in WebSphere Application Server Version 7.0 and will be removed in a future release. Re-architect your legacy applications to use javax.servlet.filter classes instead of com.ibm.servlet classes. Starting from the Servlet 2.3 specification, javax.servlet.filter classes you can intercept requests and examine responses. You can also use javax.servlet.filter classes to achieve chaining functionality, as well as embellishing or truncating responses.

HitCount application

Use the HitCount demonstration application to demonstrate how to increment a counter using a variety of methods, including:
- A servlet instance variable
- An HTTP session
- An enterprise bean

You can instruct the servlet to execute any of these methods within a transaction that you can commit or roll back. If the transaction is committed, the counter is incremented. If the transaction is rolled back, the counter is not incremented.

The enterprise bean method uses a container-managed persistence enterprise bean that persists the counter value to an Apache Derby database. This enterprise bean is configured to use the Default Datasource, which is set to the DefaultDB database.

When using the enterprise bean method, you can instruct the servlet to look up the enterprise bean, either in the WebSphere global namespace, or in the namespace local to the application.

The URL for the HitCount application is: `http://your.server.name:9080/HitCount.jsp`.

Servlets

Servlets are Java programs that use the Java Servlet Application Programming Interface (API). You must package servlets in a Web archive (WAR) file or Web module for deployment to the application server. Servlets run on a Java-enabled Web server and extend the capabilities of a Web server, similar to the way applets run on a browser and extend the capabilities of a browser.

Servlets can support dynamic Web page content, provide database access, serve multiple clients at one time, and filter data.

In the application server, discussions of servlets focus on HTTP servlets, which serve Web-based clients.

You can define servlets as welcome files. Non-servlet resources are served only when the fileServingEnabled attribute is set to `true` in the IBM extensions XMI file, ibm-web-ext.xmi, located in each Web module WEB-INF directory or by using an assembly tool to set the property in the source .war file. Serving welcome files is connected to serving static content. Therefore the fileServingEnabled attribute is set in the Web module.

Context parameters

A servlet context defines the server view of the Web application within which the servlet is running. The context also supports a servlet to access its available resourcest. Using the servlet context, a servlet can log events, obtain URL references to resources, and set and store attributes for other servlets in the context to use. These properties declare the parameters for the context of a Web application. The properties convey setup information, such as the e-mail address for the Webmaster or the name of a system with critical data.

Servlet mappings

A servlet mapping is a correspondence between a client request and a servlet. Web containers use URL paths to map client requests to servlets, and follow the URL path-mapping rules as specified in the Java Servlet specification. The container uses the Uniform Resource Identifier (URI) from the request, minus the context path, as the path to map to a servlet. The container chooses the longest matching available context path from the list of Web applications that it hosts.

JavaServer Pages

JavaServer Pages (JSP) are application components coded to the JavaServer Pages Specification. JavaServer Pages enable the separation of the Hypertext Markup Language (HTML) code from the business logic in Web pages so that HTML programmers and Java programmers can more easily collaborate in creating and maintaining pages.

JSP files support a division of roles:

HTML authors
: Develop JSP files that access databases and reusable Java components, such as servlets and beans.

Java programmers
: Create the reusable Java components and provide the HTML authors with the component names and attributes.

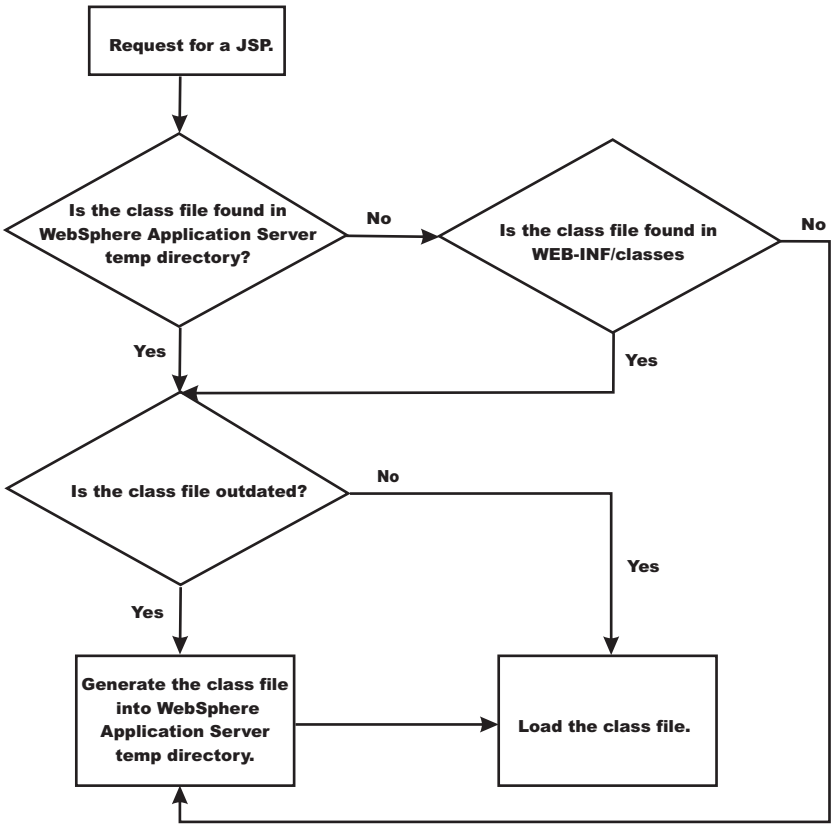
Database administrators
: Provide the HTML authors with the name of the database access and table information.

WebSphere Application Server Version 7.0 supports the JSP 2.1 specification.

JSP class file generation

At runtime, the WebSphere Application Server JavaServer Pages (JSP) engine loads JSP class files from either the WebSphere Application Server `temp` directory or a Web module's `WEB-INF/classes` directory. The JSP engine first searches for a class file in the `temp` directory and then it searches in the Web module's `WEB-INF/classes` directory.

In a default installation, the WebSphere Application Server `temp` directory is typically *`profile_root`*`/temp`. Figure 1 shows the processing logic of the JSP engine at runtime.



The batch compiler supports the generation of class files in both the WebSphere Application Server `temp` directory and a Web module's `WEB-INF/classes` directory, depending on the type of batch compiler target. In addition, the batch compiler enables the generation of class files into any directory on the filesystem, outside of the target application. Generating class files into a Web module's `WEB-INF/classes` directory enables you to deploy the Web module as a self-contained Web archive (WAR) file, or a WAR file inside an enterprise archive (EAR) file. The following table shows the batch compiler's behavior when compiling class files.

	ear.path or war.path supplied	enterpriseApp.name supplied
compileToDir not supplied; *compileToWebInf* not supplied, or is true	The class files are compiled into the Web module's `WEB-INF/classes` directory.	The class files are compiled into the Web module's `WEB-INF/classes` directory.
compileToDir not supplied; *compileToWebInf* is false	The class files are compiled into the Web module's `WEB-INF/classes` directory.	The class files are compiled into the WebSphere Application Server temp directory, usually *`profile_root`*`/temp`.
compileToDir is supplied; *compileToWebInf* not supplied, or is either true or false	The class files are compiled into the directory indicated by *compileToDir*.	The class files are compiled into the directory indicated by *compileToDir*.

Packages and directories for generated .java and .class files

By default, the .java files for all JavaServer Pages (JSP) files are generated with the package statement, `package com.ibm._jsp;`. The JSP engine's class loader knows how to load JSP classes when they are all in the same package. The .java files are located in the filesystem within a directory structure mirroring the JSP source directory structure.

If the JSP engine configuration parameter **useFullPackageNames** is set to true, the .java files are generated with the package statement

```
Package _ibmjsp.<directory structure in which the jsp is located>;
```

The usage of full package names enables the configuration of a JSP as a servlet in the `web.xml` file. See “JSP class loading settings” on page 45 for more information. The table below gives examples of packages and directory structures for generated .java and .class files.

	Java package		**Location of .java or .class files in file system**	
JSP file	**default**	**useFullPackageNames=true**	**default**	**useFullPackageNames=true**
/myJsp.jsp	com.ibm._jsp	_ibmjsp	/	/_ibmjsp
/jspFiles/ jspOne.jsp	com.ibm._jsp	_ibmjsp.jspFiles	/jspFiles	/_ibmjsp/jspFiles
/dir with spaces/jspTwo.jsp	com.ibm._jsp	_ibmjsp.dir_20_with_20_spaces	/dir with spaces	/_ibmjsp/ dir_20_with_20_spaces

Generated .java files:

When the JSP engine's **keepgenerated** configuration parameter is set to true, the `.java` file that is generated for JavaServer Pages (JSP) is retained. The `.java` file contains information that is useful in debugging.

Dependency information

In the `.java` file, immediately following the class declaration, an array of dependent files is defined, if the source JSP has any dependencies. There are three types of files that are tracked as dependencies:

1. Files that are statically included in the JSP
2. Tag files that are used by the JSP, but only tag files that are not in Java Archive (JAR) files
3. TLD files that are used by the JSP, but only TLDs that are not in JAR files

This array is always generated, but the JSP engine uses it, in determining whether a JSP needs to be recompiled, only when the trackDependencies parameter is set to true.

In the example below, three JSP fragments, one TLD and one tag file are dependencies of the JSP jsp1.jsp. There are three parts to each array entry:

1. The path to the dependency, relative to the Web module's context root. For example: `/dir1/frag1.jspf`
2. The long value representing the time the file was last modified. For example: 1082407108000
3. The String representation of the long value. For example: Mon Apr 19 16:38:28 EDT 2004

```
public final class _jsp1 extends com.ibm.ws.jsp.runtime.HttpJspBase
  implements com.ibm.ws.jsp.runtime.JspClassInformation {

  private static String[] _jspx_dependants;
  static {
  _jspx_dependants = new String[5];
  _jspx_dependants[0] = "/Banner.jspf^1082407108000^Mon Apr 19 16:38:28 EDT 2004";
  _jspx_dependants[1] = "/Footer.jspf^1077657462000^Tue Feb 24 16:17:42 EST 2004";
  _jspx_dependants[2] = "/dir1/frag1.jspf^1035396680000^Wed Oct 23 14:11:20 EDT 2002";
  _jspx_dependants[3] = "/utility.tld^1080069938000^Tue Mar 23 14:25:38 EST 2004";
  _jspx_dependants[4] = "/WEB-INF/tags/top.tag^1065440490000^Mon Oct 06 07:41:30 EDT 2003";
  }
```

Version, JSP engine options, and WEB.XML information

The generated .java source contains a comment that lists information about the file which is located at the bottom of the generated file. This information includes:

- The date and time the .java file was generated
- The version, build number and build date of the WebSphere Application Server on which the .java file was generated
- The values of the JSP engine configuration parameters that were in effect when the file was generated
- The values of any <jsp-config> elements in the web.xml file that pertained to the source JSP file.

```
/*
profile_root/installedApps/MyCell/sampleApp.ear/examples.war/WEB-INF/classes/_ibmjsp/_jsp1.java
was generated @ Wed May 03 10:05:56 EDT 2006IBM WebSphere Application Server - ND, 6.1.0.0
    Build Number: o0441.04
    Build Date: 05/01/06*******************************************************
The JSP engine configuration parameters were set as follows:

classDebugInfo =           [false]
debugEnabled =             [false]
deprecation =              [false]
compileWithAssert =        [false]
jdkSourceLevel =           [13]disableJspRuntimeCompilation =[false]
extendedDocumentRoot =     [null]
ieClassId =                [clsid:8AD9C840-044E-11D1-B3E9-00805F499D93]
keepGenerated =            [true]


outputDir =                [/QIBM/UserData/WebSphere/AppServer/V6/ND/profiles/AppSrv01/installedApps/MyCell/
                            sampleApp.ear/examples.war/WEB-INF/classes]


reloadEnabled =            [true]
reloadEnabledSet =         [true]
reloadInterval =           [5000]
trackDependencies =        [false]
usePageTagPool =           [false]
useThreadTagPool =         [true]
useImplicitTagLibs =       [true]
verbose =                  [false]
looseLibMap =              [null]
useJikes =                 [false]
useFullPackageNames =      [true]
translationContextClass =  [null]
extensionProcessorClass =  [null]
javaEncoding =             [UTF-8]
autoResponseEncoding =     [false]

*****************************************************
The following JSP Configuration Parameters were obtained from web.xml:

prelude list = [[]]
coda list = [[]]
elIgnored = [false]
pageEncoding = [null]
isXML = [false]
scriptingInvalid = [false]
*/
```

JSP class loading settings

You can configure a JavaServer Pages (JSP) class to be loaded by either the JSP engine's class loader or by the Web module's class loader.

By default, a JSP class is loaded by a unique instance of the JSP engine's class loader. The JSP engine's class loader enables reloading at runtime of a JSP class when the JSP source or one of its dependents is modified. This allows you to reload a single JSP class when necessary, without affecting any other loaded JSP classes.

JSP classes are loaded by the Web module's class loader under either of the following scenarios.

1. The JSP engine configuration parameter useFullPackageNames is set to true, and the JSP file is configured as a servlet in the web.xml file using the <servlet-class> scenario in the table below.
2. The JSP engine configuration parameters useFullPackageNames and disableJspRuntimeCompilation are both set to true. In this case, you do not need to configure a JSP file does as a servlet in the web.xml file.

Configuring JSP files as Servlets

You can configure a JSP file as a servlet in the web.xml file. There are two ways to do this. They are described in the table below.

Before you configure a JSP file as a servlet, consider the following.

1. Reloading capability - If runtime reloading of JavaServer Pages files is desired, requests for JavaServer Pages files must be handled by the JSP engine. The <servlet-class> scenario in the table below disables runtime JSP file reloading, while the <jsp-file> scenario is compatible with reloading.
2. Reducing the number of class loaders - If you do not require runtime reloading of modified JSP pages and you want to reduce the number of class loader instances, then you can use the <servlet-class> scenario in the table below. Similarly, scenario 2 in section 1 above can be used without having to configure a JSP file as a servlet.

Scenario	Example	compatible with runtime reloading	multiple class loaders used?	useFullPackageNames
<jsp-file>	<servlet> <servlet-name>jspOne</servlet-name> <jsp-file>jspOne.jsp</jsp-file> </servlet>	Yes	Yes	Can be true or false
<servlet-class>	<servlet> <servlet-name>jspTwo</servlet-name> <servlet-class>_ibmjsp.jspTwo</servlet-class> </servlet>	No	No	Must be true

The JSP batch compiler tool helps you configure JavaServer Pages files as servlets. When useFullPackageNames is true, the JSP batch compiler generates <servlet> and <servlet-mapping> elements for each JSP file that it successfully translates and compiles. The elements are written to a web.xml fragment file named generated_web.xml which is located in the binaries WEB-INF directory of a Web module processed by the JSP file batch compiler (this directory is located within the deployed application's ear file). You can copy and paste all or some of these elements into the web.xml file to configure JavaServer Pages files as servlets.

Take note of the location of the web.xml that is used by the application server. The application specific configuration is obtained from either the application binaries (the application's ear file) or from the configuration repository. If an application is deployed into WebSphere Application Server with the flag Use Binary Configuration set to true, then the WEB-INF/web.xml file is looked for in a Web module's binaries directory, not in the configuration repository. Below are examples of these two locations.

- An example of a configuration repository directory is *profile_root*/config/cells/*cellName*/applications/*enterpriseAppName*/deployments/*deployedName*/*webModuleName*

- An example of an application binaries directory is: *profile_root*/installedApps/*nodeName*/*EnterpriseAppName*/*WebModuleName*/

If the JSP batch compiler is executed on a pre-deployed application then the `web.xml` file is in the Web module's `WEB-INF` directory.

JavaServer Pages (JSP) runtime reloading settings

JavaServer Pages files can be translated and compiled at run time when the JSP file or its dependencies are modified. This is known as JSP reloading.

JSP reloading is enabled through the **reloadEnabled** JSP engine parameter in the `WEB-INF/ibm-web-ext.xmi` file:

```
<jspAttributes xmi:id="JSPAttribute_1" name="reloadEnabled" value="true"/>
```

The following table contains reload settings for production and development environments.

	Settings	
Configuration Attribute	**Production Environment**	**Development Environment**
reloadEnabled	false	true
reloadInterval	n/a (ignored if reloadEnabled is false)	approximately 5 seconds
trackDependencies	n/a (ignored if reloadEnabled is false)	true Alternatively, set this to false to improve response time if dependencies are not changing
disableJspRuntimeCompilation	true - Alternatively, set this to false if JSP files are not pre-compiled and therefore need to be compiled on the first request.	false

If the **reloadEnabled** parameter is set to true, a JSP file is reloaded at run time if the JSP file and its class file do not have the same timestamp. In addition, if **trackDependencies** is set to true then the JSP file is reloaded if the timestamp of any of its dependencies has changed since the JSP class file was last generated. If the **reloadEnabled** parameter is set to false, a JSP file is still compiled if necessary on the first request to it unless the parameter **disableJspRuntimeCompilation** is true. For example, when **disableJspRuntimeCompilation** is false and **reloadEnabled** is false, a JSP file is compiled on the first request if the class file is outdated. It would not compiled on subsequent requests even if the JSP source file is modified or the class file is deleted unless **reloadEnabled** is true

Reload interval

The reload interval is set through the **reloadInterval** JSP engine parameter:

```
<jspAttributes xmi:id="JSPAttribute_1" name="reloadInterval" value="5"/>
```

If reloading is enabled, the **reloadInterval** parameter value determines the delay between checks to see if a JSP file is outdated. For example, if **reloadInterval** is 5, the JSP engine checks to see if a JSP file is outdated only when the last such check was done more than five seconds prior to the current request for the JSP file. Once the **reloadInterval** is exceeded, reload checking is performed and the reload interval timer is reset to 0 for that JSP file. The larger the **reloadInterval**, the less frequently the JSP engine checks for the need to reload a JSP file.

Dependency tracking

Dependency tracking is set through the **trackDependencies** JSP engine parameter:

```
<jspAttributes xmi:id="JSPAttribute_1" name="trackDependencies" value="true"/>
```

If reloading is enabled, the **trackDependencies** parameter value determines whether the JSP engine tracks modifications to the requested JSP file dependencies as well as to the JSP file itself. The three types of dependencies tracked by the JSP engine are:

- files statically included in the JSP file
- tag files that are referenced in the JSP file (excluding tag files that are in JAR files)
- TLDs that are referenced in the JSP file (excluding TLDs that are in JAR files)

Dependency tracking information is always included in the generated class file even if **trackDependencies** is false. The information is not used by the JSP engine or batch compiler unless the **trackDependencies** parameter is true. This means that you can enable dependency tracking without having to recompile JSP files.

For example, the `toplevel.jsp` file statically includes the `footer.jspf` file. When the `toplevel.jsp` file is compiled, the path to the `footer.jspf` file and its timestamp are stored in the `toplevel.jsp`'s class file. As a result, the `footer.jspf` file is modified and the `toplevel.jsp` file is requested. Now that the reload interval for the `toplevel.jsp` file has been exceeded, the JSP engine compares the timestamp stored in the class file with the `footer.jspf` file timestamp on disk. Because the timestamps are different, the `toplevel.jsp` file is compiled, picking up the modification to the `footer.jspf` file. In order for dependency tracking to work, the **trackDependencies** value must be set to true at the time a JSP file is requested at run time or is processed by the batch compiler.

Disabling compilation

Disablement of run time compilation of JavaServer Pages is set via the disableJspRuntimeCompilation JSP engine parameter:

```
<jspAttributes xmi:id="JSPAttribute_1" name="disableJspRuntimeCompilation" value="true"/>
```

If the **disableJspRuntimeCompilation** parameter is set to true, the JSP engine at run time does not translate and compile JSP files; the JSP engine loads only precompiled class files. JSP source files do not need to be present in order for the class files to be loaded. With this option set to true, an application can be installed without JSP source, but must have precompiled class files. There is a Web container custom property of the same name that can be used to determine the behavior of all web modules installed in a server. If both the Web container custom property and the JSP engine option are set, the JSP engine option takes precedence. Setting the **disableJspRuntimeCompilation** parameter to true automatically sets **reloadEnabled** to false.

Reload processing sequence

The processing sequence pertaining to JSP file reloading when **trackDependencies** is false is shown in Figure 1.

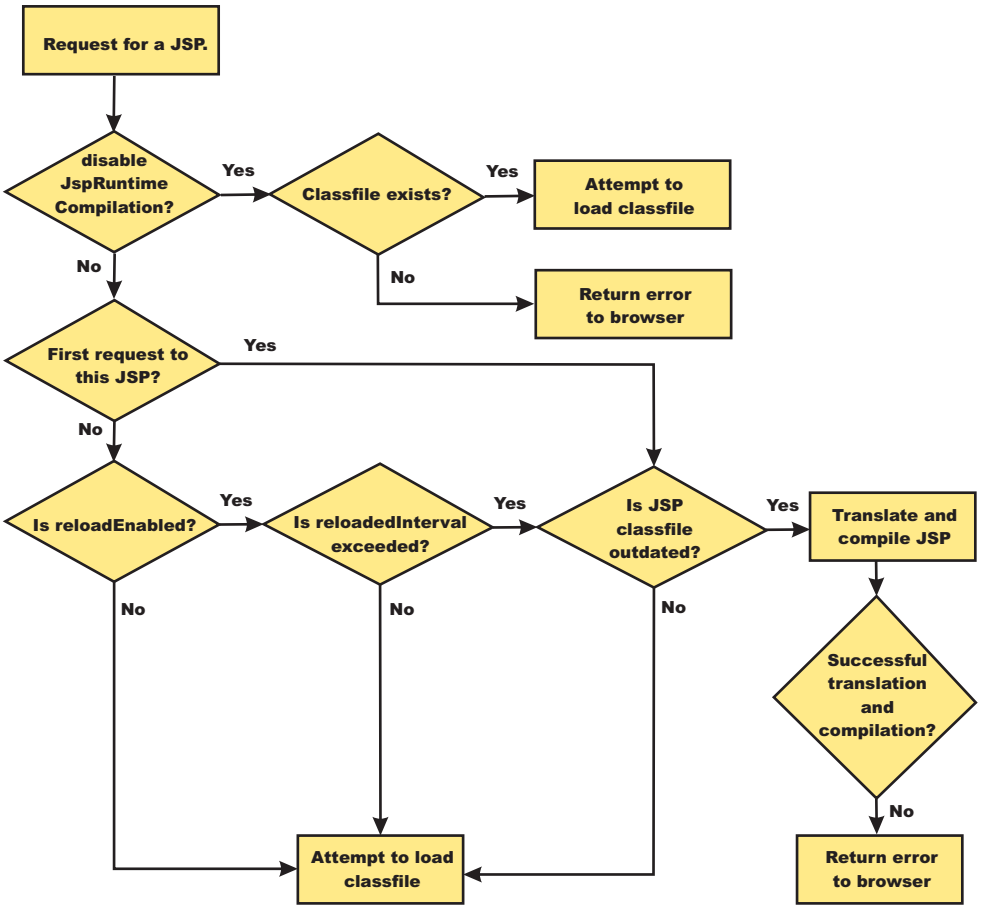


Figure 1. Reload processing sequence when **trackDependencies** *is false.*

When **trackDependencies** is true, the JSP engine does additional file system processing to determine if any of a JSP file's dependencies have changed since the JSP file was last translated and compiled. Figure 2 shows the additional processes that are performed on the 'No' path of flow chart labeled ″is JSP class file outdated?″. You can see that the path taken when disableJspRuntimeCompilation is true is the most efficient path.

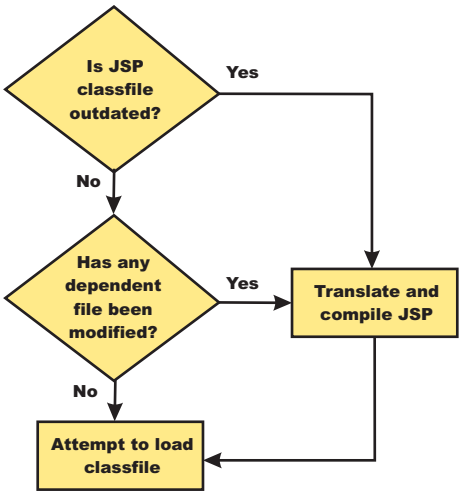


Figure 2. Additional reload processing performed when **trackDependencies** *is true.*

JSP and JSF option settings

Use this panel to configure the class reloading of Web modules such as JavaServer Pages (JSP) files and to select a JSF implementation to use with this application.

To view this administrative console panel, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **JSP and JSF options**. This panel is the same as the **Provide JSP reloading options for Web modules** panel on the application installation and update wizards.

Web module:

Specifies the name of a Web module in the installed or deployed application.

URI:

Specifies the location of the module relative to the root of the application (EAR file).

JSP enable class reloading:

Specifies whether to enable class reloading when JSP files are updated.

A Web container reloads JSP files only when the IBM extension reloadEnabled in the jspAttributes of the `ibm-web-ext.xmi` file is set to `true`.

J2EE 5 applications IBM extension files are in .xml file format. For applications versions earlier than J2EE 5, they are in the .xmi file format.

JSP reload interval in seconds:

Specifies the number of seconds to scan the application's file system for updated JSP files. The default is the value of the reloading interval attribute in the IBM extension (`META-INF/ibm-web-ext.xmi`) file of the Web module.

To enable reloading, specify a value greater than zero (for example, 1 to 2147483647). The default reload interval is 5. To disable reloading, specify zero (0). The range is from 0 to 2147483647.

The reloading interval attribute takes effect only if class reloading is enabled.

J2EE 5 applications IBM extension files are in .xml file format. For applications versions earlier than J2EE 5, they are in the .xmi file format.

Sun Reference Implementation 1.2:

Select this option to use the Sun Reference Implementation 1.2 JSF implementation. This is the default JSF implementation.

If you change the JSF implementation that you are using for your application, you must delete any previously compiled JSP files. If you precompiled your application, you must recompile. If you did not precompile, but have already requested JSP files from this application, you must delete the JSP files from the temp directory of your profile.

You can set the JSF engine configuration paramater, com.ibm.ws.jsf.JSF_IMPL_CHECK, to true to automatically mark the JSP files to recompile at application startup.

MyFaces 1.2:

Select this option to use the MyFaces 1.2 JSF implementation.

If you change the JSF implementation that you are using for your application, you must delete any previously compiled JSP files. If you precompiled your application, you must recompile. If you did not precompile, but have already requested JSP files from this application, you must delete the JSP files from the temp directory of your profile.

You can set the JSF engine configuration paramater, com.ibm.ws.jsf.JSF_IMPL_CHECK, to true to automatically mark the JSP files to recompile at application startup.

JSP run time compilation settings

By default, the JavaServer Pages (JSP) engine translates a requested JSP file, compiles the `.java` file, and loads the compiled servlet into the run time environment. You can change the JSP engine default behavior by indicating that a JSP file must not be translated or compiled at run time, even when a `.class` file does not exist.

If run time compilation is disabled, you must precompile the JSP files, which provides the following advantages:

- Reduces compilation related disk operations.
- Minimizes disk storage requirements necessary for handling temporary `.java` files generated during a run time compilation.
- Allows you to not include the JSP source files in the application.
- Allows verification that a JSP file compiled successfully before deploying and installing the application in the product

You can disable run time JSP file compilation on a global or an individual Web application basis:

- To disable the translation and compilation of JSP files for all Web applications, in the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*. Then, in the Container Settings section, click **Web container settings > Web container > Custom properties**.

  If the disableJspRuntimeCompilation property appears in the list of defined custom properties, but is set to `false`, click the property name, and then set the property to `true`.

  If this property is not included in the list of defined custom properties, click **New**, and then specify `disableJspRuntimeCompilation` in the **Name** field and `true` in the **Value** field.

  Valid settings for this property are `true` or `false`. When this property is set to `true`, translation and compilation of the JSP files is disabled at run time for all Web applications.

- To disable the translation and compilation of JSP files for a specific Web application, set the JSP engine initialization parameter disableJspRuntimeCompilation to `true`. This setting, if enabled, determines the run time behavior of the JSP engine and overrides the Web container custom property setting.

  Set this parameter through the **JavaServer Pages attribute assembly settings** page in the Assembling applications.

  Valid values for this setting are `true` or `false`. If this parameter is set to `true`, then, for that specific Web application, translation and compilation of the JSP files is disabled at run time, and the JSP engine only loads precompiled files.
- If neither the Web container custom property nor the JSP parameter is set, the first request for a JSP file results in the translation and compilation of the JSP file when the `.class` file does not exist or is outdated. Subsequent requests for the file also result in translations and compilations, but only if the following conditions are met:
  - Translations are required because the `.class` file is outdated.
  - Reloading is enabled for the Web module.
  - Reload interval is exceeded.

If you disable run time compilation and a request arrives for a JSP file that does not have a matching .class file, the JSP engine returns the following 404 error to the browser:

```
Error 404: SRVE0200E: Servlet [org.apache.jsp._jsp1]: Could not find required servlet class - _jsp1.class
```

In this case, an exception is written to the System Out (SYSOUT) and First Failure Data Capture (FFDC) logs. .

If a JSP file has a matching .class file but that file is out of date, the JSP engine still loads the .class file into memory.

Provide options to compile JavaServer Pages settings

Use this panel to specify options to be used by the JavaServer Pages (JSP) compiler.

This administrative console panel is a step in the application installation and update wizards. To view this panel, you must select **Precompile JavaServer Pages files** on the **Select Installation options** panel. Thus, to view this panel, click **Applications** → **New application** *application_path* **> Show me all installation options and parameters > Next > Next > Precompile JavaServer Pages files > Next > Step: Provide options to compile JSPs**.

You can specify the JSP compiler options on this panel only when installing or updating an application that contains Web modules. After the application is installed, you must edit the JSP engine configuration parameters of a Web module's `WEB-INF/ibm-web-ext.xmi` file to change its JSP compiler options.

Web module:

Specifies the name of a module within the application.

URI:

Specifies the location of the module relative to the root of the application (EAR file).

JSP class path:

Specifies a temporary class path for the JSP compiler to use when compiling JSP files during application installation. This class path is not saved when the application installation is complete and is not used when the application is running. This class path is used only to identify resources outside of the application which are necessary for JSP compilation and which will be identified by other means (such as shared libraries) after the application is installed. In network deployment configurations, this class path is specific to the deployment manager machine.

To specify that multiple Web modules use the same class path:
1. In the list of Web modules, select the **Select** check box beside each Web module that you want to use a particular class path.
2. Expand **Apply Multiple Mappings**.
3. Specify the desired class path.
4. Click **Apply**.

Use full package names:

Specifies whether the JSP engine generates and loads JSP classes using full package names.

When full package names are used, precompiled JSP class files can be configured as servlets in the `web.xml` file, without having to use the `jsp-file` attribute. When full package names are not used, all JSP classes are generated in the same package, which has the benefit of smaller file-system paths.

When the options useFullPackageNames and disableJspRuntimeCompilation are both `true`, a single class loader is used to load all JSP classes, even if the JSP files are not configured as servlets in the `web.xml` file.

This option is the same as the useFullPackageNames JSP engine parameter.

JDK source level:

Specifies the source level at which the Java compiler compiles JSP Java sources. Valid values are `13`, `14`, and `15`. The default value is `13`, which specifies source level 1.3.

Disable JSP runtime compilation:

Specifies whether a JSP file should never be translated or compiled at run time, even when a `.class` file does not exist.

When this option is set to `true`, the JSP engine does not translate and compile JSP files at run time; the JSP engine loads only precompiled class files. JSP source files do not need to be present in order to load class files. You can install an application without JSP source, but the application must have precompiled class files.

For a single Web application class loader to load all JSP classes, this compiler option and the **Use full package names** option both must be set to `true`.

This option is the same as the disableJspRuntimeCompilation JSP engine parameter.

JSP batch compilation

As an IBM enhancement to JavaServer Pages (JSP) support, IBM WebSphere Application Server provides a batch JSP compiler that allows JSP page compilation before application deployment. The batch compiler validates the syntax of JSP pages, translates the JSP pages into Java source files, and compiles the Java source files into Java servlet class files. The batch compiler also validates tag files and generates their Java implementation classes.

Batch compilation of JSP pages in a predeployed application simplifies the deployment process and improves the runtime performance of JSP page by eliminating first-request compilations. The batch compiler also operates on enterprise applications that have been deployed into WebSphere Application Server.

The JSP batch compiler works on Web modules that support Servlet 2.2 and later. The batch compiler works on JSP pages written to the JSP 2.1 specification or previous specifications back to JSP 1.0. It recognizes a Servlet 2.5 deployment descriptor, `web.xml`, and can use any jsp-config elements that it may

contain. In a Servlet 2.3 (JSP 1.2) or Servlet 2.2 (JSP 1.1) deployment descriptor the batch compiler recognizes and uses any taglib elements that the descriptor may contain.

Batch compiling makes the first request for a JSP page much faster because the JSP page is already translated and compiled into a servlet. Batch compiling is also useful as a fast way to resynchronize all of the JSP pages for an application.

The batch compiler supports the generation of class files in both the WebSphere Application Server `temp` directory and a Web module's `WEB-INF/classes` directory, depending on the type of batch compiler target. In addition, the batch compiler enables generation of class files into any directory on the filesystem, outside the target application. Generating class files into a Web module's `WEB-INF/classes` directory enables the Web module to be deployed as a self-contained WAR file, or a WAR inside an EAR.

Also, you can use shared libraries with the JSP batch complier. When you use the JSP batch compiler, you must either add the JAR to the WAR in the <WEB-INF>/lib directory, or add the JAR to the JVM class path to use shared libraries.

You can use the pre-touch tool for batch compilation, which can compile and load JSP class files into the application server JVM. This tool offers improved performance over the JSP batch compiler on iSeries® servers. See "Pre-touch tool for compiling and loading JSP files" on page 65 for more information.

JSPBatchCompiler command:

The batch compiler validates the syntax of JavaServer Pages, translates the JSP pages into Java source files, and compiles the Java source files into Java Servlet class files. The batch compiler also validates tag files and generates their Java implementation classes. Use this function to batch compile your JSP files and thereby enable faster responses to the initial client requests for the JSP files on your production Web server.

The batch compiler can be executed against compressed or expanded enterprise archive (EAR) files and Web application archive (WAR) files, as well as enterprise applications and Web modules that have been deployed into WebSphere Application Server. When the target is a deployed enterprise application, the server does not need to be running to execute the batch compiler. If the batch compiler is executed while the target sever is running, the server is not aware of an updated class file and does not load that class file unless the enterprise application is restarted. When the target is a compressed EAR file or WAR file, the batch compiler must expand it before executing.

Processing of Web modules

The batch compiler operates on one Web module at a time. If the target is either an EAR file or an installed enterprise application that contains more than one Web module, the batch compiler operates on each Web module individually. This is done because JSP pages are configured on a Web module basis, through the Web module's web.xml deployment descriptor file. Within a Web module, the batch compiler processes one directory at a time. It validates and translates each JSP page individually, and then invokes the Java compiler for the entire group of generated Java sources files in that directory. If one JSP page fails during the Java compilation phase, the Java compiler might not create class files for most or all of the JSP pages that successfully compiled in that directory.

JSP file extensions

The batch compiler uses four things to determine what file extensions it should process:

1. Standard JSP file extensions
   - *.jsp
   - *.jspx
   - *.jsw

- *.jsv

2. The url-pattern property of the jsp-property-group elements in the deployment descriptor file in Servlet 2.4 Web modules
3. The **jsp.file.extensions** JSP engine configuration parameter (for pre-Servlet 2.4 Web modules)
4. The batch compiler configuration parameter **jsp.file.extensions**

The standard extensions are always used by the batch compiler. If the Web module contains a Servlet 2.4 deployment descriptor, the batch compiler also processes any url-patterns found within the jsp-config element. If the batch compiler target contains the JSP engine configuration parameter **jsp.file.extensions**, then those extensions are also processed. If the batch compiler configuration parameter **jsp.file.extensions** is present, the extensions given are also processed and will override the JSP engine configuration parameter **jsp.file.extensions**.

It is a good idea to give JSP 'fragments' an extension that is not processed by the batch compiler. Statically-included fragments that do not stand alone generate translation or compilation errors if processed. The JSP 2.0 Specification suggests that you use the extension `.jspf` for such files.

Batch compiler command

The JspBatchCompiler script for running the batch compiler from the Qshell command line is found in the *app_server_root*/bin directory. An Ant task is also available for executing the batch compiler using Ant. See the topic, Batch Compiler Ant Task for additional information.

The batch compiler target is the only required parameter. The target is one of -ear.path, -war.path or -enterpriseapp.name.

```
JspBatchCompiler -ear.path | -war.path | -enterpriseapp.name <name>
   [-response.file <filename>]
   [-webmodule.name <name>]
   [-filename <jsp name | directory name>
   [-recurse <true | false>]
   [-config.root <path>]
   [-cell.name <name>]
   [-node.name <name>]
   [-server.name <name>]
   [-profileName <name>]
   [-extractToDir <path>]
   [-compileToDir <path>]
   [-compileToWebInf <true | false>]
   [-translate <true | false>]
   [-compile <true | false>]
   [-removeTempDir <true | false>]
   [-forceCompilation <true | false>]
   [-useFullPackageNames <true | false>]
   [-trackDependencies <true | false>]
   [-createDebugClassfiles <true | false>]
   [-keepgenerated <true | false>]
   [-keepGeneratedclassfiles <true | false>]
   [-usePageTagPool <true | false>]
   [-useThreadTagPool <true | false>]
   [-classloader.parentFirst <true | false>]
   [-classloader.singleWarClassloader <true | false>]
   [-additional.classpath <classpath to additional JAR files and classes>]

   [-verbose <true | false>]
   [-deprecation <true | false>]
   [-javaEncoding <encoding>
   [-jdkSourceLevel <13 | 14 | 15>]
   [-compilerOptions <space-separated list of java compiler options>]
   [-useJikes <true | false>]
   [-jsp.file.extensions <file extensions to process>]
   [-log.level <SEVERE | WARNING | INFO | CONFIG | FINE | FINER | FINEST | OFF>]
```

```
*** See batchcompiler.properties.default in app_server_root/bin for more information. ***
*** See JspCBuild.xml in app_server_root/bin for information about the public WebSphere Ant task JspC. ***
```

The batch compiler is aware of three groups of configuration parameters:

1. JSP engine configuration parameters for a Web module.

   See the topic, JSP engine configuration parameters.
2. Batch compiler response file configuration parameters.

   These are the parameters that are found in a batch compiler response file. See -response.file, below.
3. Batch compiler command line configuration parameters.

   These are the parameters entered on the command line when running the batch compiler.

The batch compiler looks at all three groups of configuration parameters in order to determine which value for a parameter is used when compiling JSP pages. When resolving the value for a given parameter, the precedence is:

1. If the parameter is found on the command line, its value is used.
2. If the parameter is not found on the command line, the batch compiler looks for the parameter in a response file named on the command line.
3. If no response file is named, or if the parameter is not found therein, the batch compiler looks for the parameter in the Web module's JSP engine configuration parameters.

If a configuration parameter is not found among these three groups, then a default value is used. The default values for the configuration parameters are given below along with the description of the parameters.

With one exception, these parameters are not case sensitive; -profileName is case sensitive. If the values specified for these arguments are comprised of two or more words separated by spaces, you must add quotation marks around the values.

The batch compiler does not create, or set the values of, equivalent JSP engine parameters. This means that if a JSP page in a deployed Web module is modified and is recompiled by the JSP engine at run time, the JSP engine's configuration parameters will determine the engine's behavior. For example, if you use the batch compiler to compile a Web module and you use the -useFullPackageNames true option, the JSP files will be compiled to support that option. But the JSP engine parameter useFullPackageNames must also be set to true in order for the JSP runtime to be able to load the compiled JSP pages. If JSP pages are modified in a deployed Web module, then the engine's parameters should be set to the same values used in batch compilation.

To use the JSP batch compiler, enter one of the following commands on a single line at an operating system command prompt.:

- **ear.path | war.path | enterpriseapp.name**

  Represents the full path to a single compressed or expanded enterprise application archive (EAR) file or Web application archive (WAR) file, or the name of the deployed enterprise application that you want to compile. For example:
  - `JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear`
  - `JspBatchCompiler -war.path /home/wasuser/wars/examples.war`
  - `JspBatchCompiler -enterpriseapp.name myEnterpriseApp -webmodule.name my.war -filename mydirectory/main.jsp`
- **response.file**

  Specifies the path to a file that contains configuration parameters used by the batch compiler. The *response.file* is used only if it is given on the command line; it is ignored if it is present in a response file.

  The template response file, batchcompiler.properties.default, is found in the *app_server_root*/`properties` directory. Copy this template to create your own response files containing defaults for the parameters in

which you are interested. You can configure all the required and optional parameters, except the response.file in a response file.**For example:** JspBatchCompiler -response.file /home/wasuser/myproject/batchc.props

Default : null

- **webmodule.name**

  Represents the name of the specific Web module that you want to batch compile. If this argument is not set, all Web modules in the enterprise application are compiled. This parameter is used only when *ear.path* or *enterpriseapp.name* is given. This parameter is useful when JSP pages in a specific Web module within a deployed enterprise application need to be regenerated, because all shared library dependencies are picked up.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -webmodule.name myWebModule.war

  Default: All Web modules in an EAR file or enterprise application are compiled if this parameter is not given.

- **filename**

  Represents the name of a single JSP file that you want to compile. If this argument is not set, all files in the Web module are compiled. Alternatively, if *filename* is set to the name of a directory, only the JSP files in that directory and that directory's child directories are complied. The name is relative to the context root of the Web module.

  Example 1: If you want to compile the file, `myTest.jsp`, and it is found in `/subdir/myJSPs`, you would enter `-filename /subdir/myJSPs/myTest.jsp`.

  Example 2: If you want to compile all JSP files in `/subdir/myJSPs` and its child directories, you would enter `-filename subdir/myJSPs`.

  Default: All JSP files in the Web module are compiled. Entering `-filename /` is equivalent to the default.

- **recurse**

  Determines whether subdirectories beneath the target directory are processed. This parameter is used only when the *filename* parameter is given. Set value to `false` to process only the directory named *filename* parameter; and not its subdirectories.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -filename /subdir1 -recurse false.

  Default: true; All directories beneath the target directory are processed.

- **config.root**

  Specifies the location of the WebSpehere Application Server configuration directory. This parameter is used only when *enterpriseapp.name* is given.

  Default: *profile_root*`/config`

- **cell.name**

  Specifies the name of the cell in which the application is deployed. This parameter is used only when *enterpriseapp.name* is given.

  Default: The default is obtained from the profile script that is used. The symbolic name of this variable is WAS_CELL.

- **node.name**

  Specifies the name of the node in which the application is deployed. This parameter is used only when *enterpriseapp.name* is given.

  Default: The default is obtained from the profile script that is used. The symbolic name of this variable is WAS_NODE.

- **server.name**

  Represents the name of the server in which the application is deployed. This parameter is used only when *enterpriseapp.name* is given.

  Default: server1

- **profileName**

  Specifies the name of the profile you want to use. This parameter is used only when `enterpriseapp.name` is given.

Example: JspBatchCompiler -enterpriseApp.name sampleApp -profileName AppServer-3

Default: The default profile is used. This is obtained from the `setupCmdLine` script in the *app_server_root*`/bin` directory. The symbolic name is DEFAULT_PROFILE_SCRIPT.

- **extractToDir**

  Specifies the directory into which predeployed enterprise archive (EAR) files and Web application archive (WAR) files will be extracted before the batch compiler operates on them. This parameter is ignored when *enterpriseapp.name* is given. The extractToDir parameter is used as described in the table below.

  Example: JspBatchCompiler -ear.path c:\myproject\sampleApp.ear -extractToDir /home/wasuser/mytempdir.

  Use-case: You must extract a compressed archive before it is batch compiled. You can also extract an expanded archive to a new directory as well. In both cases, extraction leaves the original archive untouched, which may be useful while development is underway.

  Default values:

	Expanded archive	**Compressed archive**
extractToDir supplied	The batch compiler extracts the archive to extractToDir before operating on it. If a file or directory of the same name as the archive already exists in the extractToDir, the batch compiler removes the archive completely before extracting that archive. If the batch compiler exits with no errors, it compresses the archive in place in the extractToDir, even if the original EAR file or WAR file was expanded. If errors are encountered during compilation, the EAR file or WAR file is left in the expanded state even if the original EAR file or WAR file was compressed.	
extractToDir not supplied	The batch compiler operates on the EAR file or WAR file in place (does not extract it to another directory) and the archive remains expanded after the batch compiler finishes.	The batch compiler extracts the archive to the directory returned by the JVM property ″java.io.tmpdir″. The rest of the behavior described above, when extractToDir is supplied, is the same in this case.

  The default is `server1`.

- **compileToDir**

  Specifies the directory into which JSP pages are translated into Java source files and compiled into class files. This directory can be anywhere on the filesystem, but the batch compiler's default behavior is usually adequate. The batch compiler's behavior when compiling class files is described in the table below

  Example:: JspBatchCompiler -enterpriseApp.name sampleApp -compileToDir /home/wasuser/myTargetDir

  Use-case: This parameter enables you to generate the Java and class files into a directory outside of the target, which may be useful if you want to compare the newly generated files with their previous versions which remain untouched within the target.

  Default values:

	ear.path or war.path supplied	**enterpriseApp.name supplied**
compileToDir not supplied; compileToWebInf not supplied, or is true	The class files are compiled into the Web module's `WEB-INF/classes` directory	The class files are compiled into the Web module's `WEB-INF/classes` directory.
compileToDir not supplied; compileToWebInf is false	The class files are compiled into the Web module's `WEB-INF/classes` directory.	The class files are compiled into the WebSphere Application Server temp directory (usually *profile_root*`/temp`).

compileToDir is supplied; compileToWebInf not supplied, or is either true or false	The class files are compiled into the directory indicated by compileToDir.	The class files are compiled into the directory indicated by compileToDir.

- **compileToWebInf**

  Specifies whether the target directory for the compiled JSP class files should be the Web module's `WEB-INF/classes directory`. This parameter is used only when *enterpriseApp.name* is given, and it is overridden by *compileToDir* if *compileToDir* is given.

  The batch compiler's default behavior is to compile to the Web module's `WEB-INF/classes` directory. The batch compiler's behavior when compiling class files is described in the table above.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -compileToWebInf false.

  Use-case: Set this parameter to false when enterpriseApp.name is supplied and you want the class files to be compiled to the WebSphere Application Server temp directory instead of the Web module's `WEB-INF/classes` directory. Recommendation: if this parameter is set to false, set forceCompilation to true if there are any JSP class files in the `WEB-INF/classes` directory.

  Default: true; see the table above.
- **forceCompilation**

  Specifies whether the batch compiler is forced to recompile all JSP resources regardless or whether the JSP page is outdated.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -forceCompilation true.

  Use-case: Especially useful when creating an archive for deployment, to make sure all JSP classes are up to date.

  Default: false
- **useFullPackageNames**

  Specifies whether the batch compiler generates full package names for JSP classes. The default is to generate all JSP classes in the same package. The JSP engine's class loader knows how to load JSP classes when they are all in the same package. The default has the benefit of generating smaller file-system paths. Full package names have the benefit of enabling the configuration of precompiled JSP class files as servlets in the `web.xml` file without use of the `jsp-file` attribute, resulting in a single class loader (the Web application's class loader) being used to load all such JSP classes. Similarly, when the JSP engine's configuration attributes **useFullPackageNames** and **disableJspRuntimeCompilation** are both true, a single class loader is used to load all JSP classes, even if the JSP pages are not configured as servlets in the `web.xml` file.

  When useFullPackageNames is set to true, the batch compiler generates a file called `generated_web.xml` in the Web module's `WEB-INF` directory. This file contains servlet configuration information for each JSP page that is successfully translated and compiled. The information can optionally be copied into the Web module's `web.xml` file so that the JSP pages are loaded as servlets by the Web container. Note that if a JSP page is configured as a servlet in this way, no reloading of the JSP page is done at run time if the JSP page is modified. This is because the JSP page is treated as a regular servlet and requests for it do not pass through the JSP engine.

  Example: JspBatchCompiler –enterpriseApp.name sampleApp –useFullPackageNames true

  Use-case: Enables JSP classes to be loaded by a single class loader.

  Default: false
- **removeTempDir**

  Specifies whether the Web module's temp directory is removed. The batch compiler by default generates JSP class files into a Web module's `WEB-INF/classes` directory. JSP class files are generated into the `temp` directory at run time if a JSP page is modified and JSP reloading is enabled. By batch compiling all the JSP pages in a Web module and also removing the `temp` directory, disk resources are preserved. You can only use the removeTempDir parameter when -enterpriseApp.name is given.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -removeTempDir true.

  Use-case: Free up disk space by clearing out a Web application's `temp` directory.

Default: false

- **translate**

  Specifies whether JSP pages are translated and compiled. Set translate to false if you do not want JSP pages to be translated and compiled. You must use this option in conjunction with `-removeTempDir` to tell the batch compiler to remove only the `temp` directory and to do no further processing.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -translate false -removeTempDir true.

  Use-case: Free up disk space by clearing out a Web application's `temp` directory, without invoking JSP processing.

  Default: true

- **compile**

  Specifies whether JSP pages go through the Java compilation phase. Set compile to `false` if you do not want JSP pages to go through the Java compilation phase.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -compile false

  Use-case: If you only want JSP pages to be syntax-checked, set -compile to false. You can set -keepgenerated to true if you want to see the `.java` files that are generated during the translation phase.

  Default: true

- **trackDependencies**

  Specifies whether the batch compiler recompiles a JSP page when any of its dependencies have changed, even if the JSP page itself has not changed. Tracking dependencies incurs a significant runtime performance penalty because the JSP Engine checks the filesystem on every request to a JSP page to see if any of its dependencies have changed. The dependencies tracked by WebSphere Application Server are :

  1. Files statically included in the JSP page
  2. Tag files used by the JSP page (excluding tag files that are in JAR files)
  3. TLD files used by the JSP page (excluding TLD files that are in JAR files)

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -trackDependencies true.

  Use-case: Useful in a development environment.

  Default: false

- **createDebugClassfiles**

  Specifies whether the batch compiler generates class files that contain SMAP information, as per JSR 45, **Debugging support for Other Languages**.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -createDebugClassfiles true

  Use-case: Use this parameter when you want to be able to debug JSP pages in your JSR 45-compliant IDE.

  Default: false

- **keepgenerated**

  Specifies whether the batch compiler saves or erases the generated Java source files created during the translation phase.

  If set to `true`, WebSphere Application Server saves the generated `.java` files used for compilation on your server. By default, this argument is set to `false` and the `.java` files are erased after the class files have compiled.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -keepgenerated true

  Use-case: Use this parameter when you want to review the Java code generated by the batch compiler.

  Default: false

- **keepGeneratedclassfiles**

  Specifies whether the batch compiler saves or erases the class files generated during the compilation of Java source files.

Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -keepGeneratedclassfiles false -keepgenerated false

Use-case: Set this parameter to false if you only want to see if there are any translation or compilation errors in your JSP pages. If paired with -keepgenerated false, this parameter results in all generated files being removed before the batch compiler completes.

Default: true

- **usePageTagPool**

  Enables or disables the reuse of custom tag handlers on an individual JSP page basis.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -usePageTagPool true

  Use-case: Use this parameter to enable JSP-page-based reuse of tag handlers.

  Default: false
- **useThreadTagPool**

  Enables or disables the reuse of custom tag handlers on a per request thread basis per Web module.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -useThreadTagPool true

  Use-case: Use this parameter to enable Web module-based reuse of tag handlers.

  Default: false
- **classloader.parentFirst**

  Specifies the search order for loading classes by instructing the batch compiler to search the parent class loader prior to application class loader. This parameter is only used when *ear.path* or *enterpriseApp.name* is given.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -classloader.parentFirst false

  Use-case: Set this parameter to false when your Web module contains a JAR file that is also found in the server lib directory, and you want your Web module's JAR file to be picked up first.

  Default: true
- **classloader.singleWarClassloader**

  Specifies whether to use one class loader per enterprise archive (EAR) file or one class loader per Web application archive (WAR) file. Used only when *ear.path* or *enterpriseApp.name* is given.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -classloader.singleWarClassloader true

  Use-case: Set this parameter to true when a Web module depends on JAR files and classes in another Web module in the same enterprise application.

  Default: false; One class loader is created per WAR file with no visibility of classes in other Web modules.
- **additional.classpath**

  Specifies additional class path entries to be used when parsing and compiling JSP pages. This parameter is used only when `war.path` is given. When `war.path` is the target, WebSphere Shared Libraries are not picked up by the batch compiler. Therefore, if your WAR file relies on, for example, a JAR file that is configured in WebSphere Application Server as a shared library, then use this option to point to that JAR file. In addition, if you give `war.path` and also use the `-extractToDir` parameter, then any JAR files that are in the WAR file's manifest class-path is not added to the class path (since the WAR file has now been extracted by itself outside the EAR file in which it resides). Use `-additional.classpath` in this case to point to the necessary JAR files. Add the full path to needed resources, separated by your system-dependent path separator.

  Example: JspBatchCompiler -war.path /home/wasuser/myproject/examples.war -additional.classpath /home/wasuser/myJars/someJar.jar:/home/wasuser/myClasses

  Use-case: Use this parameter to add to the class path JAR files and classes outside of your WAR file. At run time, these same JAR files and classes have to be made available through the standard WebSphere Application Server configuration mechanisms.

Default: null

- **verbose**

  Specifies whether the batch compiler should generate verbose output while compiling the generated sources.

  Example: JspBatchCompiler -war.path /home/wasuser/myproject/examples.war -verbose true

  Use-case: Set this parameter to true when you want to see Java compiler class loading and other messages.

  Default: false

- **deprecation**

  Indicates the compiler should generate deprecation warnings while compiling the generated sources.

  Example: JspBatchCompiler -war.path /home/wasuser/myproject/examples.war -deprecation true

  Use-case: Set this parameter to true when you want to see Java compiler deprecation messages.

  Default: false

- **javaEncoding**

  Specifies the encoding that will be used when the .java file is generated, and when it is compiled by the Java compiler. When -javaEncoding is set, that encoding is passed to the java compiler via the -encoding argument. Note that encoding is not supported by Jikes.

  Example: JspBatchCompiler -war.path/home/wasuser/myproject/examples.war -javaEncoding Shift-JIS

  Use-case: Set this parameter when the page encoding of your JSP pages is not UTF-8 compatible.

  Default value: UTF-8.

- **jdkSourceLevel**

  This is a new JSP engine parameter which was introduced in WebSphere Application Server version 6.1 to support JDK 5. This parameter should be used instead of the compileWithAssert parameter, although compile WithAssert still works in version 6.1.

  The default value for this parameter is 13. This parameter requires regeneration of Java source. The following are jdkSourceLevel paramater values:

  – **13 (default)** - This value will disable all new language features of JDK 1.4 and JDK 5.0.
  – **14** - This value will enable the use of the assertion facility and will disable all new language features of JDK 5.0.
  – **15** - This value will enable the use of the assertion facility and all new language features of JDK 5.0.

  Example: JspBatchCompiler -war.path c:\myproject\examples.war -jdkSourceLevel 14

  Use-case: Set this parameter when you want to enable or disable the language features of JDK 1.4 and JDK 5.0

  Default value: 13

- **compilerOptions**

  Specifies a list of strings to be passed on the Java compiler command. This is a space-separated list of the form ″arg1 arg2 argn″.

  Example: JspBatchCompiler -war.path /home/wasuser/myproject/examples.war -compilerOptions ″ -bootclasspath <path>″

  Use-case: Use this parameter if you need Java compiler arguments other than verbose, deprecation and Assert facility support.

  Default: null

- **useJikes**

  Specifies whether Jikes should be used for compiling Java sources. NOTE: Jikes is not shipped with WebSphere Application Server.

  Example: JspBatchCompiler -ear.path /home/wasuser/myproject/sampleApp.ear -useJikes true

  Use-case: Set this parameter to true in order for the batch compiler to use Jikes as the Java compiler.

Default value: false

- **jsp.file.extensions**

  Specifies the file extensions to be processed by the batch compiler. This is a semicolon- or colon-separated list of the form ″*.ext1;*.ext2:*.extn″. Note that this parameter is not necessary for Servlet 2.4 Web applications because the url-pattern property of the jsp-property-group elements in the deployment descriptor can be used to identify extensions that should be treated as JSP pages.

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -jsp.file.extensions *jspz;*.jspt

  Use-case: Use this parameter to add additional extensions to the be processed by the batch compiler.

  Default: null. See section, ″JSP file extensions″, in this topic for additional information.

- **log.level**

  Specifies the level of logging that is directed to the console during batch compilation. Values are SEVERE | WARNING | INFO | CONFIG | FINE | FINER | FINEST | OFF

  Example: JspBatchCompiler -enterpriseApp.name sampleApp -log.level FINEST

  Use-case: Set this parameter higher or lower to control logging output. FINEST generates the most output useful for debugging.

  Default: CONFIG

Batch compiler ant task:

The ant task **JspC** exposes all the batch compiler configuration options. It executes the batch compiler under the covers. It is backward compatible with the WebSphere Application Server 5.x version of the **JspC** ant task. The following table lists all the ant task attribute and their batch compiler equivalents.

JspC attribute	Equivalent batch complier parameter
earPath	-ear.path
warPath	-war.path
src Same as warPath, for backward compatiblity	-war.path
enterpriseAppName	-enterpriseapp.name
responseFile	-response.file
webmoduleName	-webmodule.name
fileName	-filename -config.root
configRoot	-config.root
cellName	-cell.name
nodeName	-node.name
serverName	-server.name
profileName	-profileName
extractToDir	-extractToDir
compileToDir same as compileToDir, for backward compatibility	-compileToDir -compileToDir
compileToWebInf	-compileToWebInf
compilerOptions	-compilerOptions
recurse	-recurse
removeTempDir	-removeTempDir
translate	-translate

compile	-compile
forceCompilation	-forceCompilation
useFullPackageNames	-useFullPackageNames
trackDependencies	-trackDependencies
createDebugClassfiles	-createDebugClassfiles
keepgenerated	-keepgenerated
keepGeneratedclassfiles	-keepGeneratedclassfiles
usePageTagPool	-usePageTagPool
useThreadTagPool	-useThreadTagPool
classloaderParentFirst	-classloader.parentFirst
classloaderSingleWarClassloader	-classloader.singleWarClassloader
additionalClasspath	-additional.classpath
classpath same as additionalClasspath, for backward compatibility	-additional.classpath
verbose	-verbose
deprecation	-deprecation
javaEncoding	-javaEncoding
compileWithAssert	-compileWithAssert
useJikes	-useJikes
jspFileExtensions	-jsp.file.extensions
logLevel	-log.level
wasHome	none
Classpathref	none
jdkSourceLevel	-jdkSourceLevel

Below is an example of a build script with multiple targets, each with different attributes. The following commands are used to execute the script:

On Windows:

```
ws_ant -Dwas.home=%WAS_HOME% -Dear.path=%EAR_PATH% -Dextract.dir=%EXTRACT_DIR%
ws_ant jspc2 -Dwas.home=%WAS_HOME% -Dapp.name=%APP_NAME% -Dwebmodule.name=%MOD_NAME%
ws_ant jspc3 -Dwas.home=%WAS_HOME% -Dapp.name=%APP_NAME% -Dwebmodule.name=%MOD_NAME% -Ddir.name=%DIR_NAME%
```

On UNIX® or i5/OS®:

```
ws_ant -Dwas.home=$WAS_HOME -Dear.path=$EAR_PATH -Dextract.dir=$EXTRACT_DIR
ws_ant jspc2 -Dwas.home=$WAS_HOME -Dapp.name=$APP_NAME -Dwebmodule.name=$MOD_NAME
ws_ant jspc3 -Dwas.home=$WAS_HOME -Dapp.name=$APP_NAME -Dwebmodule.name=$MOD_NAME -Ddir.name=$DIR_NAME
```

Example build.xml Using the JspC Task

```
<project name="JSP Precompile" default="jspc1" basedir=".">
 <taskdef name="wsjspc" classname="com.ibm.websphere.ant.tasks.JspC"/>
 <target name="jspc1" description="example using  a path to an EAR, and extracting the EAR to a directory">
  <wsjspc wasHome="${was.home}"
            earpath="${ear.path}"
            forcecompilation="true"
            extractToDir="${extract.dir}"
            useThreadTagPool="true"
            keepgenerated="true"
```

```
 />
</target>
<target name="jspc2" description="example using an enterprise app and webmodule">
 <wsjspc wasHome="${was.home}"
           enterpriseAppName="${app.name}"
           webmoduleName="${webmodule.name}"
           removeTempDir="true"
           forcecompilation="true"
           keepgenerated="true"

 />
</target>
<target name="jspc3" description="example using an enterprise app, webmodule and specific directory">
 <wsjspc wasHome="${was.home}"
           enterpriseAppName="${app.name}"
           webmoduleName="${webmodule.name}"
           fileName="${dir.name}"
           recurse="false"
           forcecompilation="true"
           keepgenerated="true"

 />
</target>
</project>
```

Pre-touch tool for compiling and loading JSP files:

When enabled, the pre-touch mechanism causes all JavaServer Pages (JSP) files to be compiled within the Web module for which they are configured. You can also configure some or all JSP files to be class loaded and JIT-compiled.

To enable the pre-touch mechanism, use Rational Application Developer to specify the following JSP attributes, which are Assembly Property Extensions for your Web module:

- **prepareJSPs (Required)**

  When this attribute is present, all JSP files are compiled at application server startup. This activity runs in a separate thread, allowing the application server to finish other startup actions in parallel. The numeric attribute value represents the minimum size in kilobytes that a JSP file must be in order to also be class loaded and JIT-compiled. The default is 0, which causes all JSP files to be class loaded and JIT-compiled.

  Note: JSP file compilation is different from JIT compilation. JSP compilation generates bytecodes, whereas JIT translates the bytecodes into machine code at run time.
- **prepareJSPAttribute (Optional)**

  The pre-touch mechanism compiles and JIT-compiles JSP files by directly invoking the JSP service method, thus making the JSP file susceptible to incurring exceptions because it is called out of context. Such exceptions are avoided by immediately checking the value of this attribute, causing a quick exit from the service method when the JSP was prepared by this tool. This attribute value is added as a request parameter and is composed of alphanumeric characters that your JSP files do not expect to use during normal initiation.
- **prepareJSPThreadCount (Optional)**

  Set this numeric attribute to the number of threads that you would like this mechanism to start up to compile your JSP files. Since a thread makes use of just one processor, multi-processor systems may better utilize this pre-touch mechanism by specifying a value greater than 1. The default setting for this attribute is 1, representing the number of threads that are created to perform pre-touch processing for this Web module.
- **prepareJSPClassload (Optional)**

Set this attribute to either a whole number or the word *changed*. By entering *changed*, only those JSP filess that have been updated or not previously touched, for example, those JSP files that need to be converted from a .jsp file to a .java file, are class loaded. By entering a numerical value, for example, 1000, the pre-touch tool starts class loading at the 1000th JSP that it processes and all subsequent JSP files. This is convenient in the event that the application server is stopped when starting the pre-touch tool. You can then check the server logs to see how many JSP files have been processed and update the prepareJSPClassload value accordingly to avoid duplicating work. If a JSP file is not class loaded, it cannot be JIT compiled. As a result, if a JSP file does not satisfy the requirements of the prepareJSPClassload attribute, but satisfies the requirements of the prepareJSPs attribute, the JSP file is compiled if it has been updated, but is not class loaded or JIT compiled.

Batch compiler class path:

The batch compiler builds its class path as shown in the table below. When the batch compiler target is a Web archive (WAR) file and `war.path` is supplied, the configuration *additional.classpath* parameter is used to give extra class path information.

	Batch compiler target		
Location added to class path	enterpriseapp.name	ear.path	war.path
WebSphere Application Server JAR files and classes	yes	yes	yes
JAR files listed in manifest class path for a Web module	yes	yes	yes, when the target WAR is inside an EAR and *–extractToDir* is not used; otherwise, no.
Shared libraries	yes	no	no
Web module JAR files and classes	yes	yes	yes
additional.classpath parameter to batch compiler	no	no	yes

Global tag libraries (deprecated)

JavaServer Pages (JSP) tag libraries contain classes for common tasks such as processing forms and accessing databases from JSP files.

Tag libraries encapsulate, as simple tags, core functionality common to many Web applications. The Java Standard Tag Library (JSTL) supports common programming tasks such as iteration and conditional processing, and provides tags for:

- manipulating XML documents
- supporting internationalization
- using Structured Query Language (SQL)

Tag libraries also introduce the concept of an expression language to simplify page development, and include a version of the JSP expression language.

A tag library has two parts - a Tag Library Descriptor (TLD) file and a Java archive (JAR) file.

tsx:dbconnect tag JavaServer Pages syntax (deprecated):

Use the <tsx:dbconnect> tag to specify information needed to make a connection to a database through Java DataBase Connectivity (JDBC) or Open Database Connectivity (ODBC) technology.

Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

The <tsx:dbconnect> syntax does not establish the connection. Use the <tsx:dbquery> and <tsx:dbmodify> syntax instead to reference a <tsx:dbconnect> tag in the same JavaServer Pages (JSP) file to establish the connection.

When the JSP file compiles into a servlet, the Java processor adds the Java coding for the <tsx:dbconnect> syntax to the servlet service() method, which means a new database connection is created for each request for the JSP file.

This section describes the syntax of the <tsx:dbconnect> tag.

```
<tsx:dbconnect id="connection_id"
    userid="db_user" passwd="user_password"
    url="jdbc:subprotocol:database"
    driver="database_driver_name"
    jndiname="JNDI_context/logical_name">
</tsx:dbconnect>
```

where:

- **id**

  Represents a required identifier. The scope is the JSP file. This identifier is referenced by the connection attribute of a <tsx:dbquery> tag.
- **userid**

  Represents an optional attribute that specifies a valid user ID for the database that you want to access. Specify this attribute to add the attribute and its value to the request object.

  Although the userid attribute is optional, you must provide the user ID. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this information in the JSP file.
- **passwd**

  Represents an optional attribute that specifies the user password for the userid attribute. (This attribute is not optional if the userid attribute is specified.) If you specify this attribute, the attribute and its value are added to the request object.

  Although the passwd attribute is optional, you must provide the password. See <tsx:userid> and <tsx:passwd> for an alternative to hard coding this attribute in the JSP file.
- **url** and **driver**

  Respresents a required attribute if you want to establish a database connection. You must provide the URL and driver.

  The application server supports connection to JDBC databases and ODBC databases.
  - For a JDBC database, the URL consists of the following colon-separated elements: jdbc, the subprotocol name, and the name of the database to access. An example for a connection to the Sample database included with IBM DB2 is:

    ```
    url="jdbc:db2:sample"
    driver="com.ibm.db2.jdbc.app.DB2Driver"
    ```
  - For an ODBC database, use the Sun JDBC-to-ODBC bridge driver included in their Java2 Software Developers Kit (SDK) or another vendor's ODBC driver.

    The url attribute specifies the location of the database. The driver attribute specifies the name of the driver to use in establishing the database connection.

    If the database is an ODBC database, you can use an ODBC driver or the Sun JDBC-to-ODBC bridge. If you want to use an ODBC driver, refer to the driver documentation for instructions on specifying the database location with the url attribute and the driver name.

    If you use the bridge, the url syntax is `jdbc:odbc:`*`database`*. An example follows:

    ```
    url="jdbc:odbc:autos"
    driver="sun.jdbc.odbc.JdbcOdbcDriver"
    ```

    Note: To enable the application server to access the ODBC database, use the ODBC Data Source Administrator to add the ODBC data source to the System DSN configuration. To access the ODBC Administrator, click the ODBC icon on the Windows NT® Control Panel.
- **jndiname**

Represents an optional attribute that identifies a valid context in the application server Java Naming and Directory Interface (JNDI) naming context and the logical name of the data source in that context. The Web administrator configures the context using an administrative client such as the WebSphere Administrative Console.

If you specify the jndiname attribute, the JSP processor ignores the driver and url attributes on the <tsx:dbconnect> tag.

An empty element (such as <url></url>) is valid.

dbquery tag JavaServer Pages syntax (deprecated):

Use the <tsx:dbquery> tag to establish a connection to a database, submit database queries, and return the results set.

Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

The <tsx:dbquery> tag does the following:
1. References a <tsx:dbconnect> tag in the same JavaServer Pages (JSP) file and uses the information the tag provides to determine the database URL and driver. You can also obtain the user ID and password from the <tsx:dbconnect> tag if those values are provided in the <tsx:dbconnect> tag.
2. Establishes a new connection
3. Retrieves and caches data in the results object.
4. Closes the connection and releases the connection resource.

This section describes the syntax of the <tsx:dbquery> tag.

```
<%-- SELECT commands and (optional) JSP syntax can be placed within the tsx:dbquery. --%>
<%-- Any other syntax, including HTML comments, are not valid. --%>
<tsx:dbquery id="query_id" connection="connection_id" limit="value" >
</tsx:dbquery>
```

where:
- **id**

  Represents the identifier of this query. The scope is the JSP file. Use `id` to reference the query. For example, from the <tsx:getProperty> tag, use `id` to display the query results.

  The `id` is a tsx reference to the bean and can be used to retrieve the bean from the page contect. For example, if id is named mySingleDBBean, instead of using:
  – if (mySingleDBBean.getValue(″UISEAM″,0).startsWith(″N″))

  use:
  – com.ibm.ws.webcontainer.jsp.tsx.db.QueryResults bean = (com.ibm.ws.webcontainer.jsp.tsx.db.QueryResults)pageContext. findAttribute(″mySingleDBBean″); if (bean.getValue(″UISEAM″,0).startsWith(″N″)). . .

  The bean properties are dynamic and the property names are the names of the columns in the results set. If you want different column names, use the SQL keyword for specifying an alias on the SELECT command. In the following example, the database table contains columns named FNAME and LNAME, but the SELECT statement uses the `AS` keyword to map those column names to FirstName and LastName in the results set:

  ```
  Select FNAME, LNAME AS FirstName, LastName from Employee where FNAME='Jim'
  ```
- **connection**

  Represents the identifier of a <tsx:dbconnect> tag in this JSP file. The <tsx:dbconnect> tag provides the database URL, driver name, and optionally, the user ID and password for the connection.
- **limit**

Represents an optional attribute that constrains the maximum number of records returned by a query. If this attribute is not specified, no limit is used. In such a case, the effective limit is determined by the number of records and the system caching capability.

- SELECT command and JSP syntax

  Represents the only valid SQL command, SELECT. The <tsx:dbquery> tag must return a results set. Refer to your database documentation for information about the SELECT command. See other articles in this section for a description of JSP syntax for variable data and inline Java code.

dbmodify tag JavaServer Pages syntax (deprecated):

The <tsx:dbmodify> tag establishes a connection to a database and then adds records to a database table.

Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

The <tsx:dbmodify> tag does the following:

1. References a <tsx:dbconnect> tag in the same JavaServer Pages (JSP) file and uses the information provided by that tag to determine the database URL and driver.

   Note: You can also obtain the user ID and password from the <tsx:dbconnect> tag if those values are provided in the <tsx:dbconnect> tag.
2. Establishes a new connection.
3. Updates a table in the database.
4. Closes the connection and releases the connection resource.

This section describes the syntax of the <tsx:dbmodify> tag.

```
<%-- Any valid database update commands can be placed within the DBMODIFY tag. -->
<%-- Any other syntax, including HTML comments, are not valid. -->
<tsx:dbmodify connection="connection_id">
</tsx:dbmodify>
```

where:

- **connection**

  Represents the identifier of a <tsx:dbconnect> tag in this JSP file. The <tsx:dbconnect> tag provides the database URL, driver name, and (optionally) the user ID and password for the connection.
- Database commands

  Represents valid database commands. Refer to your database documentation for details

In the following example, a new employee record is added to a database. The values of the fields are based on user input from this JavaServer Pages (JSP) file and referenced in the database commands using the <tsx:getProperty> tag.

```
<tsx:dbmodify connection="conn" >
insert into EMPLOYEE
    (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT,EDLEVEL)
values
('<tsx:getProperty name="request" property=request.getParameter("EMPNO") />',
'<tsx:getProperty name="request" property=request.getParameter("FIRSTNME") />',
'<tsx:getProperty name="request" property=request.getParameter("MIDINIT") />',
'<tsx:getProperty name="request" property=request.getParameter("LASTNAME") />',
'<tsx:getProperty name="request" property=request.getParameter("WORKDEPT") />',
<tsx:getProperty name="request" property=request.getParameter("EDLEVEL") />)
</tsx:dbmodify>
```

tsx:getProperty tag JavaServer Pages syntax and examples (deprecated):

The <tsx:getProperty> tag gets the value of a bean to display in a JavaServer Pages (JSP) file.

Note: Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

This IBM extension of the Sun JSP <jsp:getProperty> tag implements all of the <jsp:getProperty> function and adds the ability to introspect a database bean created using the IBM extension <tsx:dbquery> or <tsx:dbmodify>.

Note: You cannot assign the value from this tag to a variable. The value, generated as output from this tag, displays in the browser window.

This section describes the syntax of the <tsx:getProperty> tag:

```
<tsx:getProperty name="bean_name"
  property="property_name" />
```

where:

- **name**

  Represents the name of the bean declared by the id attribute of a <tsx:dbquery> syntax within the JSP file. See <tsx:dbquery> for an explanation. The value of this attribute is case-sensitive.
- **property**

  Represents the property of the bean to access for substitution. The value of the attribute is case-sensitive and is the locale-independent name of the property.

Tag example:

```
<tsx:getProperty name="userProfile" property="username" />
```

tsx:userid and tsx:passwd tag JavaServer Pages syntax (deprecated):

With the <tsx:userid> and <tsx:passwd> tags, you do not have to hard code a user ID and password in the <tsx:dbconnect> tag.

Note: Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

Use the <tsx:userid> and <tsx:passwd> tags to accept user input for the values and then add that data to the request object. You can access the request object with a JavaServer Pages (JSP) file, such as the *JSPEmployee.jsp* example that requests the database connection.

You must use <tsx:userid> and <tsx:passwd> tags within a <tsx:dbconnect> tag.

This section describes the syntax of the <tsx:userid> and <tsx:passwd> tags.

```
<tsx:dbconnect id="connection_id"
    <font color="red"><userid></font>
    <tsx:getProperty name="request" property=request.getParameter("userid") />
    <font color="red"></userid></font>
    <font color="red"><passwd></font>
    <tsx:getProperty name="request" property=request.getParameter("passwd") />
    <font color="red"></passwd></font>
    url="protocol:database_name:database_table"
    driver="JDBC_driver_name">
</tsx:dbconnect>
```

where:

- **<tsx:getProperty>**

  Represents the syntax as a mechanism for embedding variable data.

- **userid**

  Represents a reference to the request parameter that contains the user ID. You must add the parameter to the request object that passes to this JSP file. You can set the attribute and its value in the request object, using an HTML form or a URL query string to pass the user-specified request parameters.
- **passwd**

  Represents a reference to the request parameter that contains the password. Add the parameter to the request object that passes to this JSP file. You can set the attribute and its value in the request object, using an HTML form or a URL query string, to pass user-specified values.

tsx:repeat tag JavaServer Pages syntax (deprecated):

The <tsx:repeat> tag repeats a block of HTML tagging.

Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

Use the <tsx:repeat> syntax to iterate over a database query results set. The <tsx:repeat> syntax iterates from the start value to the end value until one of the following conditions is met:
- The end value is reached.
- An exception is thrown.

If an exception of the types **ArrayIndexOutOfBoundsException** or **NoSuchElementException** is created before a block completes, output is written only for the iterations up to and not including the iteration during which the exception was created. All other exceptions results in no output being written for that tag instance.

This section describes the syntax of the <tsx:repeat> tag:

```
<tsx:repeat index="name" start="starting_index" end="ending_index">
</tsx:repeat>
```

where:
- **index**

  Represents an optional name used to identify the index of this repeat block. The scope of the index is NESTED. Its type must be integer.
- **start**

  Represents an optional starting index value for this repeat block. The default is 0.
- **end**

  Represents an optional ending index value for this repeat block. The maximum value is 2,147,483,647.

  If the value of the end attribute is less than the value of the start attribute, the end attribute is ignored.

Combining tsx:repeat and tsx:getProperty JavaServer Pages tags (deprecated)

Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

The following code snippet shows you how to code these tags:

```
<tsx:repeat>
<tr>
    <td><tsx:getProperty name="empqs" property="EMPNO" />
    <tsx:getProperty name="empqs" property="FIRSTNME" />
    <tsx:getProperty name="empqs" property="WORKDEPT" />
```

```
    <tsx:getProperty name="empqs" property="EDLEVEL" />
    </td>
</tr>
</tsx:repeat>
```

Example: Using tsx:repeat JavaServer Pages tag to iterate over a results set (deprecated):

The <tsx:repeat> tag iterates over a results set. The results set is contained within a bean. The bean can be a static bean, for example, a bean created by using the IBM WebSphere Studio database wizard, or a dynamically generated bean, for example, a bean generated by the <tsx:dbquery> syntax.

Note: Support for tsx tags in the JavaServer Pages (JSP) engine are deprecated in WebSphere Application Server Version 6.0. Instead of using the tsx tags, you should use equivalent tags from the JavaServer Pages Standard Tag Library (JSTL).

The following table is a graphic representation of the contents of a bean called, *myBean*:

	col1	col2	col3
row0	friends	Romans	countrymen
row1	bacon	lettuce	tomato
row2	May	June	July

Some observations about the bean:

- The column names in the database table become the property names of the bean. The <tsx:dbquery> section describes a technique for mapping the column names to different property names.
- The bean properties are indexed. For example, `myBean.get(Col1(row2))` returns `May`.
- The query results are in the rows. The <tsx:repeat> tag iterates over the rows, beginning at the start row.

The following table compares using the <tsx:repeat> tag to iterate over a static bean, versus a dynamically generated bean:

Static Bean Example	<tsx:repeat> Bean Example

myBean.class
`// Code to get a connection`
 	
`// Code to get the data`
`Select * from myTable;`
 	
`// Code to close the connection`
 	
JSP file
`<tsx:repeat index=abc>`
`<tsx:getProperty name="myBean"`
`property="col1(abc)" />`
`</tsx:repeat>`
 	
Notes®:
• The bean (myBean.class) is a static bean.
• The method to access the bean properties is myBean.get(*property*(*index*)).
• You can omit the property index, in which case the index of the enclosing <tsx:repeat> tag is used. You can also omit the index on the <tsx:repeat> tag.
• The <tsx:repeat> tag iterates over the bean properties row by row, beginning with the start row.
	JSP file
	`<tsx:dbconnect id="conn"`
	`userid="alice"passwd="test"`
	`url="jdbc:db2:sample"`
	`driver="COM.ibm.db2.jdbc.app.DB2Driver">`
	`</tsx:dbconnect >`
	
	`<tsx:dbquery id="dynamic"`
	`connection="conn" >`
	`Select * from myTable;`
	`</tsx:dbquery>`
	
	`<tsx:repeat index=abc>`
	`<tsx:getProperty name="dynamic"`
	`property="col1(abc)" />`
	`</tsx:repeat>`
	
	Notes:
	• The bean (dynamic) is generated by the <tsx:dbquery> tag and does not exist until the syntax executes.
	• The method to access the bean properties is dynamic.getValue(*"property"*, *index*).
	• You can omit the property index, in which case the index of the enclosing <tsx:repeat> tag is used. You can also omit the index on the <tsx:repeat> tag.
	• The <tsx:repeat> tag syntax iterates over the bean properties row by row, beginning with the start row.

Implicit and explicit indexing

Examples 1, 2, and 3 show how to use the <tsx:repeat> tag. The examples produce the same output if all indexed properties have 300 or fewer elements. If there are more than 300 elements, Examples 1 and 2 display all elements, while Example 3 shows only the first 300 elements.

Example 1 shows *implicit indexing* with the default start and default end index. The bean with the smallest number of indexed properties restricts the number of times the loop repeats.

```
<table>
<tsx:repeat>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" />
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="address" />
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="telephone" />
  </tr></td>
</tsx:repeat>
</table>
```

Example 2 shows indexing, starting index, and ending index:

```
<table>
<tsx:repeat index=myIndex start=0 end=2147483647>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=city(myIndex) />
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=address(myIndex) />
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property=telephone(myIndex) />
</tr></td>
</tsx:repeat>
</table>
```

Example 3 shows *explicit indexing* and ending index with implicit starting index. Although the index attribute is specified, you can still implicitly index the indexed property city because the (`myIndex`) tag is not required.

```
<table>
<tsx:repeat index=myIndex end=299>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="city" /t>
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="address(myIndex)" />
  </tr></td>
  <tr><td><tsx:getProperty name="serviceLocationsQuery" property="telephone(myIndex)" />
  </tr></td>
</tsx:repeat>
</table>
```

Nesting <tsx:repeat> blocks

You can nest <tsx:repeat> blocks. Each block is separately indexed. This capability is useful for interleaving properties on two beans, or properties that have subproperties. In the example, two <tsx:repeat> blocks are nested to display the list of songs on each compact disc in the user's shopping cart.

```
<tsx:repeat index=cdindex>
  <h1><tsx:getProperty name="shoppingCart" property=cds.title /></h1>
  <table>
  <tsx:repeat>
    <tr><td><tsx:getProperty name="shoppingCart" property=cds(cdindex).playlist />
    </td></tr>
  </tsx:repeat>
  </table>
 </tsx:repeat>
```

JavaServer Pages migration best practices and considerations

The standard JavaServer Pages (JSP) tags from JSP 1.1 such as jsp:include, jsp:useBean, and <%@ page %>, a will migrate successfully to JSP 2.0. However, there are several areas that must be considered when migrating JavaServer Pages. This topic discusses the areas that you must consider when migrating JavaServer Pages.

Classes from the unnamed or default package

As of JSP 2.0, referring to any classes from the unnamed or default package is not allowed. This can result in a translation error on some containers, specifically those that run in a JDK 1.4 or greater environment which will also break compatibility with some older JSP applications. However, as of JDK 1.4, importing classes from the unnamed package is not valid. See Java 2 Platform, Standard Edition Version 1.4.2 Compatibility with Previous Releases for details. Therefore, for forwards compatibility, applications must not rely on the unnamed package. This restriction also applies for all other cases where classes are referenced, such as when specifying the class name for a tag in a Tag Library Descriptor (TLD) file.

Page encoding for JSP documents

There have been noticeable differences in internationalization behavior on some containers as a result of ambiguity in the JSP 1.2 specification. However, steps were taken to minimize the impact on backwards compatibility and overall, the internationalization abilities of JSP files have been greatly improved.

In JSP specification versions prior to JSP 2.0, JSP pages in XML syntax, JSP documents, and those in standard syntax determined their page encoding in the same fashion, by examining the pageEncoding or contentType attributes of their page directive, defaulting to ISO-8859-1 if neither was present.

As of JSP 2.0, the page encoding for JSP documents is determined as described in section 4.3.3 and appendix F.1 of the XML specification, and the pageEncoding attribute of those pages is only checked to make sure it is consistent with the page encoding determined as per the XML specification. As a result of

this change, JSP documents that rely on their page encoding to be determined from their pageEncoding attribute are no longer decoded correctly. These JSP documents must be changed to include an appropriate XML encoding declaration.

Additionally, in JSP 1.2, page encodings are determined on translation unit basis whereas in JSP 2.0, page encodings are determined on the basis of each file. Therefore, if the a.jsp file statically includes the b.jsp file, and a page encoding is specified in the a.jsp file but not in the b.jsp file, in JSP 1.2 the encoding for the a.jsp file is used for the b.jsp file, but in JSP 2.0, the default encoding is used for the b.jsp file.

web.xml file version

The JSP container uses the version of the web.xml file to determine whether you are running a JSP 1.2 application or a JSP 2.0 application. Various features can behave differently depending on the version of the web.xml file. The following is a list of things JSP developers should be aware of when upgrading their web.xml file from version Servlet 2.3 to version Servlet 2.4:

1. EL expressions are ignored by default in JSP 1.2 applications. When you upgrade a Web application to JSP 2.0, EL expressions are interpreted by default. You can use the escape sequence \$ to escape EL expressions that should not be interpreted by the container. Alternatively, you can use the isELIgnored page directive attribute, or the <el-ignored> configuration element to deactivate EL for entire translation units. Users of JSTL 1.0 must upgrade their taglib imports to the JSTL 1.1 uris or use the _rt versions of the tags, for example, use c_rt instead of c or fmt_rt instead of fmt.
2. Web applications that contain files with an extension of .jspx will have those files interpreted as JSP documents, by default. You can use the JSP configuration element <is-xml> to treat .jspx files as regular JSP pages, but there is no way to disassociate .jspx from the JSP container.
3. The escape sequence \$ was not reserved in JSP 1.2. The output for any template text or attribute value that appeared as \$ in JSP 1.2 was \$, however, the output now is just $.

jsp:useBean tag

WebSphere Application Server version 5.1 and later enforces more strict adherence to the specification for the jsp:useBean tag: with type and class attributes. Specifically, you should use the type attribute should be used to specify a Java type that cannot be instantiated as a JavaBean. For example, a Java type that is an abstract class, interface, or a class with no public no-args constructor. If the class attribute is used for a Java type that cannot be instantiated as a JavaBean, the WebSphere Application Server JSP container produces a unrecoverable translation error at translation time.

Generated packages for JSP classes

Any reliance on generated packages for JSP classes will result in non-portable JSP files. Packages for generated classes are implementation-specific and therefore you should not rely on these packages.

JspServlet class

Any reliance on the existence of a JspServlet class will cause unrecoverable error problems. WebSphere Application Server version 6.0 and later no longer uses a JspServlet class.

JavaServer Pages specific Web container custom properties

You can configure name-value pairs of data, where the name is a property key and the value is a string value that you can use to set internal system configuration properties. Defining a new property enables you to configure a setting beyond what is available in the administrative console. The following is a list of the available JavaServer Pages custom properties. The JavaServer Pages custom properties are case sensitive.

com.ibm.wsspi.jsp.allowjspoutputelementmismatch:

CTS requirements in previous releases was not applicable to the product, therefore the JSP container supported multiple occurrences of properties in the jsp:output element. In the current release, CTS compliance requires that the JSP container strictly enforces rules about multiple occurrences of properties in the jsp:output element. You can use the *com.ibm.wsspi.jsp.allowjspoutputelementmismatch* custom property to relax the enforcement of the rule for backward compatibility.

Name	Value
com.ibm.wsspi.jsp.allowjspoutputelementmismatch	*true*

com.ibm.wsspi.jsp.allowtaglibprefixusebeforedefinition:

CTS compliance requires that a tag library directive that defines a prefix must occur before that prefix is used in a custom tag. Because CTS requirements in previous releases were not required, this rule was not enforced; however, you can use the *com.ibm.wsspi.jsp.allowtaglibprefixusebeforedefinition* custom property to relax the enforcement of the rule for backward compatibility.

Name	Value
com.ibm.wsspi.jsp.allowtaglibprefixusebeforedefinition	*true*

com.ibm.wsspi.jsp.allowtaglibprefixredefinition:

CTS compliance requires that if a tag library prefix is already defined with a different URI within a JSP, the product must create a translation error. Because CTS requirements in previous releases were not required, this rule was not enforced; however, you can use the *com.ibm.wsspi.jsp.allowtaglibprefixredefinition* custom property to relax the enforcement of the rule for backward compatibility.

Name	Value
com.ibm.wsspi.jsp.allowtaglibprefixredefinition	*true*

com.ibm.wsspi.jsp.allowunmatchedendtag:

In version 5 of the product, improper termination of end tags was ignored whereas in version 6, a translation exception is created. This change of behavior in version 6 causes problems to users who are migrating their applications, which had improperly terminated end tags, to version 6. In version 6, to facilitate migration, a WebContainer property, com.ibm.wsspi.jsp.allowunmatchedendtag, and a JSPAttribute, allowUnmatchedEndTag),are provided. Enabling these properties provides the version 5 behavior.

Name	Value
com.ibm.wsspi.jsp.allowunmatchedendtag	*true*

com.ibm.wsspi.jsp.userepeatint:

The custom property *com.ibm.wsspi.jsp.userepeatint* ....

Name	Value
com.ibm.wsspi.jsp.userepeatint	*true*

com.ibm.wsspi.jsp.usescriptvardupinit:

The code generated for a JSP file assumed that the same tag variables to be declared twice in an If-Else condition even if the variable had a 'page' scope. The custom property *com.ibm.wsspi.jsp.usescriptvardupinit* is used to enable this feature for all the applications deployed on a particular server. If the compatibility feature is required only for a specific application then the JSPAttribute useScriptVarDupInit needs to be enabled. If both the options are set then the JSPAttribute takes preference over the webcontainer property.

Name	Value
com.ibm.wsspi.jsp.usescriptvardupinit	*true*

com.ibm.wsspi.jsp.usestringcast:

The *com.ibm.wsspi.jsp.usestringcast* property explicitly adds a 'String cast' to the relative path before inclusion when you migrate version 5.1 applications. In version 6.0.x, the generated Java source for the JSP file did not add the ″implicit″ cast, for return types of type String when the request.getAttribute method is called. When a JSP file includes a resource whose relative path does not evaluate to a String, then the include fails as the include takes only a String as a relative path of the resource. You can set the com.ibm.wsspi.jsp.usestringcast custom property, which would affect all the deployed applications or you can set the property as a JSPAttribute, useStringCast, in the extensions file, which affects only the application for which the property is set.

Name	Value
com.ibm.wsspi.jsp.usestringcast	*true*

com.ibm.wsspi.jsp.usecdatatrim:

You can use the *com.ibm.wsspi.jsp.usecdatatrim* custom property to trim the text before creating CDATA section which eliminates the extra whitespaces addedUsers Affected: WebSphere Application server version 6 users who are migrating their applications from version 5 to version 6 and whose JSPs make use of nesting tags in separate lines. Problem Description: JSP files that make use of nesting tags have extra lines in the code generated for this section. In version 6, you could eliminate the extra spaces by appending all the lines into one as in: '' The extra line is added in the Java code generated because the text is not trimmed before creating the CDATA section. Therefore, a switch is provided to enable the trimming of the text before creating the CDATA section. You can set the JSP attribute, useCDataTrim, at the Web module level or set the *com.ibm.wsspi.jsp.usecdatatrim* Web container property at the Web container level using the following name-value pair.

Name	Value
com.ibm.wsspi.jsp.usecdatatrim	

com.ibm.ws.jsp.getparameterreturnemptystring:

This property is for returning an empty string for actions not set in a JSP file. If a JSP file contains an action and that property has not been set, the JSP engine returns null on WebSphere Application Server versions 6.0 and 6.1. However, in WebSphere Application Server version 5.1 an empty string is returned. This custom property was added to provide backwards compatibility. When it is set to true, the value returned on a call to jsp:getProperty is an empty string instead of null for versions 6.0 and 6.1.

Name	Value
com.ibm.ws.jsp.getparameterreturnemptystring	true

com.ibm.wsspi.jsp.evalquotedandescapedexpression:

This property is for compiling functions that contain an expression. The JSP translation code was modified to handle escape characters and quotations properly when determining whether to evaluate an expression or to treat it as a literal string. To apply this behavior globally across all Web applications, add the following name-value pair as a Web container custom property.

Name	Value
com.ibm.wsspi.jsp.evalquotedandescapedexpression	true

To enable this new behavior for a single application, you must also add the evalquotedandescapedexpression JSP attribute to the ibm-web-ext.xmi file of the failing application and set the value to `true`. An example entry is:

```
<jspAttributes xmi:id="JSPAttribute_1" name="evalquotedandescapedexpression" value="true"/>
```

The attribute id value must be unique.

com.ibm.ws.jsp.jdksourcelevel:

This property is for setting the JDK source level through the administrative console. A JSP engine parameter can be configured for different levels of JDK, but this requires that you set the jdksourcelevel JSP attribute in the Web extension file for each Web module. However, you can use the com.ibm.ws.jsp.jdkjdksourcelevel custom property to set the JSP attribute globally using the Web container custom property. If this attribute is also defined in the Web extension file, the property defined in the Web extension file supersedes the custom property for that particular application. This custom property is not case sensitive. The default value is 13.

Name	Value
com.ibm.ws.jsp.jdksourcelevel	13, 14, or 15

com.ibm.wsspi.jsp.recompilejsponrestart:

This property forces JSP files that were compiled at runtime to be recompiled every time the application is restarted. This is helpful if you switch the underlying JSF implementation. This property is best used on development environments.

Name	Value
com.ibm.wsspi.jsp.recompilejsponrestart	true

com.ibm.wsspi.jsp.modifyPageContextVariable:

During the translation phase of a tag file that is compiled, the JSP container implicitly uses the pageContext variable for the PageContext object. The use of the pageContext variable as an implicit variable name in tag files does not comply with the JSP Specification.

If compilation errors occur for applications that use a local pageContext variable in their tag file, set the com.ibm.wsspi.jsp.modifyPageContextVariable custom property to`true` to remove the use of the pageContext variable name in the generated Java code for tag files.

Name	Value
com.ibm.wsspi.jsp.modifyPageContextVariable	true

User profiles and authorities

WebSphere Application Server uses two OS/400® user profiles by default, QEJB and QEJBSVR.

The QEJB user profile is shipped as part of the operating system. This user profile is used only when accessing validation list objects used for storing the encoded passwords used with WebSphere Application Server. For more information on using validation list objects to store encoded passwords, see Restoring or replacing damaged validation list objects

The QEJBSVR user profile is created on your iSeries server when you install WebSphere Application Server. This profile is the default profile under which all application servers run. Directories and files used by WebSphere Application Server are normally owned by user profile QEJBSVR. The WebSphere Application Server runtime, administration tools, and Qshell scripts sets the ownership and authorities correctly on any objects created. If you create objects manually outside of the WebSphere Application Server tools, or if you modify the authorities on objects used by WebSphere Application Server, you must ensure QEJBSVR has the correct authorities to these objects.

You can also use the grtwasaut script and the rvkwasaut script to modify authorities on integrated file system objects. When you create new directories for WebSphere Application Server, the QEJBSVR user profile must have read and execute authorities (*RX) to those directories.

If you have specified another user profile to run your application servers, it is recommended that you specify QEJBSVR for its group profile. See Running application servers under specific user profiles for more information.

Web application deployment troubleshooting tips

Deployment of a Web application is successful if you can access the application by typing a Uniform Resource Locator (URL) in a browser or if you can access the application by following a link. If you cannot access your application, follow these steps to eliminate some common errors that can occur during migration or deployment.

Web module does not run in WebSphere Application Server Version 5.x or 6.x

Symptom Your Web module does not run when you migrate it to Version 5.x or 6.x

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Versions 5.x and 6.x, you must use class loader policies to set visibility.

Recommended response Reassemble an existing module, or change the visibility settings in the class loader policies.

See Class loaders and Class loading for more information.

Web module does not run in WebSphere Application Server Version 6.1

Symptom Your Web module does not run when you migrate it to Version 6.1.

Problem In Version 4.x, the classpath setting that affected visibility was *Module Visibility Mode*. In Version 6.1, you must use class loader policies to set visibility.

Recommended response Reassemble an existing module, or change the visibility settings in the class loader policies.

See Class loaders and Class loading for more information.

Welcome page is not visible.

Symptom You cannot access an application with a Web path of:

```
/webapp/myapp
```

Problem The default welcome page for a Web application is assumed to be *index.html*. You cannot access the default page of the *myapp* application unless it is named *index.html*.

Recommended response To identify a different welcome page, modify the properties of the Web module during assembly, see the topic, Assembling Web applications for more information in the *Developing and deploying applications* PDF book.

HTML files are not found.

Symptom — Your Web application ran successfully on prior versions, but now you encounter errors that the welcome page (typically *index.html*), or referenced HTML files are not found:

```
Error 404: File not found: Banner.html
Error 404: File not found: HomeContent.html
```

Problem — For security and consistency reasons, Web application URLs are now case-sensitive on all operating systems.

Suppose the content of the index page is as follows:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 5.0 Frameset//EN">
<HTML>
<TITLE>
Insurance Home Page
</TITLE>
    <frameset   rows="18,80">
    <frame      src="Banner.html"           name="BannerFrame"  SCROLLING=NO>
    <frame      src="HomeContent.html"      name="HomeContentFrame">
    </frameset>
</HTML>
```

However the actual file names in the `\WebSphere\AppServer\installedApps\...` directory where the application is deployed are:

```
banner.html
homecontent.html
```

Recommended response — To correct this problem, modify the *index.html* file to change the names *Banner.html* and *HomeContent.html* to *banner.html* and *homecontent.html* to match the names of the files in the deployed application.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Web applications: Resources for learning

Use the following links to find relevant supplemental information about Web applications. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming model and decisions

- J2EE BluePrints for Web applications
- Redbook on the design and implementation of Servlets, JSP files, and enterprise beans

Programming instructions and examples

- WebSphere Studio Application Developer Programming Guide
- Sun's Java™ Tutorial on Servlets and JavaServer Pages

- Web delivered samples in the Samples Gallery

Programming specifications
- Java 2 Software Development Kit (SDK)
- Servlet 2.4 Specification
- JavaServer Pages 2.0 Specification
- Differences between JavaScript and ECMAScript
- ISO 8859 Specifications
- Java 2 Platform, Standard Edition (J2SE)

Configuring session management by level

When you configure session management at the Web container level, all applications and the respective Web modules in the Web container normally inherit that configuration, setting up a basic default configuration for the applications and Web modules below it. However, you can set up different configurations individually for specific applications and Web modules that vary from the Web container default. These different configurations override the default for these applications and Web modules only.

About this task

Note: When you overwrite the default session management settings on the application level, all the Web modules below that application inherit this new setting unless they too are set to overwrite these settings.

1. Open the administrative console.
2. Select the level that this configuration applies to:
   - For the Web container level:
     a. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container** → **Session management**.
   - For the enterprise application level:
     a. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Session management**.
   - For the Web module level:
     a. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Manage Modules**.
     b. Select a Web module from the list of Web modules defined for this application.
     c. Click **> Session Management**.
3. Manage sessions by configuring session tracking, session timeouts and serializing access.
4. If you are working on the Web module or application level and want these settings to override the inherited Session Management settings, under **General Properties**, select **Override session management**.
5. Click **Apply** and **Save**.

Session management settings

Use this page to manage HTTP session support. This support includes specifying a session tracking mechanism, setting maximum in-memory session count, controlling overflow, and configuring session timeout.

To view this administrative console page at the Web container level, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management** .

Note that the session management settings can be overridden at the application level.

Session tracking mechanism
Specifies a mechanism for HTTP session management.

Mechanism	Function	Default
Enable SSL ID Tracking	**Note:** This feature is deprecated in WebSphere Application Server version 7.0. You can reconfigure session tracking to use cookies or modify the application to use URL rewriting. Specifies that session tracking uses Secure Sockets Layer (SSL) information as a session ID. Enabling SSL tracking takes precedence over cookie-based session tracking and URL rewriting. There are two parameters available if you enable SSL ID tracking: SSLV3Timeout and Secure Authentication Service (SAS). SSLV3Timeout specifies the time interval after which SSL sessions are renegotiated. This is a high setting and modification does not provide any significant impact on performance. The SAS parameter establishes an SSL connection only if it goes out of the Java Virtual Machine (JVM) to another JVM. If all the beans are co-located within the same JVM, the SSL used by SAS does not hinder performance. These are set by editing the `sas.server.properties` and `sas.client.props` files located in the *product_installation_root*\properties directory, where *product_installation_root* is the directory where WebSphere Application Server is installed. **Note:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.	9600 seconds

Enable cookies	Specifies that session tracking uses cookies to carry session IDs. If cookies are enabled, session tracking recognizes session IDs that arrive as cookies and tries to use cookies for sending session IDs. If cookies are not enabled, session tracking uses Uniform Resource Identifier (URL) rewriting instead of cookies (if URL rewriting is enabled). Enabling cookies takes precedence over URL rewriting. Click **Enable cookies** to change these settings. Application level session management settings override the server level session management settings. Because session management is defined at the application level, enabling cookies for the administration console is handled in the deployment.xml file.
Enable URL rewriting	Specifies that the session management facility uses rewritten URLs to carry the session IDs. If URL rewriting is enabled, the session management facility recognizes session IDs that arrive in the URL if the encodeURL method is called in the servlet.
Enable protocol switch rewriting	This option is only available when **Enable URL rewriting** is selected. This option specifies that the session ID is added to a URL when the URL requires a switch from HTTP to HTTPS or from HTTPS to HTTP. If rewriting is enabled, the session ID is required to go between HTTP and HTTPS.

Maximum in-memory session count

Specifies the maximum number of sessions to maintain in memory.

The meaning differs depending on whether you are using in-memory or distributed sessions. For in-memory sessions, this value specifies the number of sessions in the base session table. Use the Allow overflow property to specify whether to limit sessions to this number for the entire session management facility or to allow additional sessions to be stored in secondary tables. For distributed sessions, this value specifies the size of the memory cache for sessions. When the session cache has reached its maximum size and a new session is requested, the session management facility removes the least recently used session from the cache to make room for the new one.

Note: Do not set this value to a number less than the maximum thread pool size for your server.

Allow overflow

Specifies that the number of sessions in memory can exceed the value specified by the Max in-memory session count property. This option is valid only in non-distributed sessions mode.

Session timeout
Specifies how long a session can go unused before it is no longer valid. Specify either `Set timeout` or `No timeout`. Specify the value in minutes greater than or equal to two.

The value specified in a Web module deployment descriptor file takes precedence over the administrative console settings. However, the value of this setting is used as a default when the session timeout is not specified in a Web module deployment descriptor. Note that to preserve performance, the invalidation timer is not accurate to the second. When the write frequency is time based, ensure that this value is least twice as large as the write interval.

Security integration
Specifies that when security integration is enabled, the session management facility associates the identity of users with their HTTP sessions

Serialize session access
Specifies that concurrent session access in a given server is not allowed.

Maximum wait time	Specifies the maximum amount of time a servlet request waits on an HTTP session before continuing execution. This parameter is optional and expressed in seconds. The default is 5 seconds. Under normal conditions, a servlet request waiting for access to an HTTP session gets notified by the request that currently owns the given HTTP session when the request finishes.
Allow access on timeout	Specifies whether the servlet is started normally or aborted in the event of a timeout. If this box is checked, the servlet is started normally. If this box is not checked, the servlet execution aborts and error logs are generated.

Configuring session tracking

About this task

Review the Session tracking options to plan your approach to session management. To configure session tracking, complete the following:

1. Go to the appropriate level of Session Management.
2. Specify the session tracking mechanism that you want to pass the session ID between the browser and the servlet:
   - To track sessions with cookies, click **Enable Cookies**.

     To change the cookie settings, click **Modify**.
   - To track sessions with URL rewriting, click **Enable URL Rewriting**.

     If you want to enable protocol switch rewriting, click **Enable protocol switch rewriting.**
   - To track sessions with SSL information, click **Enable SSL ID tracking**.

     Note: Session tracking using the SSL ID is deprecated in WebSphere Application Server version 7.0. You can reconfigure session tracking to use cookies or modify the application to use URL rewriting.
3. Click **Apply**.
4. Click **Save**.

Serializing access to session data

The Servlet API supports concurrent access to a session in a given server instance. WebSphere Application Server provides an option to prevent the concurrent access to a session in a given server instance so that concurrent modification of a session does not occur in a given server instance.

About this task

Preventing concurrent access to a session is achieved by synchronizing the requests based on session. When this feature is turned on, a session is obtained for the request before invoking the servlet and requests are synchronized by locking the session for the servlet initiation time. Note that synchronization is based on the memory copy of session. So this feature cannot serialize requests across servers based on session when session affinity fails.

You can also use the serializing access to session data feature to synchronize session objects inside of servlets or JavaServer pages. Applications cannot synchronize session objects inside of their servlets or JavaServer Pages because a deadlock with the session manager may occur. The deadlock occurs because the session manager does not expect the use of more than one locking mechanism. You can ensure that only one request can access the session at a time through the use of the configuration option, Allow serial access.

Use this feature only when concurrent modification of the same session data is possible and is not desirable by the application. This feature has overhead of serializing the requests based on a session.

Do the following to synchronize session access:

1. Select the level of Session Management on which you want to serialize session access.
2. Under Serialize Session access, click **Allow serial access**.
3. In the Maximum wait time box, type the amount of time, in milliseconds, a servlet waits on a session before continuing execution. The default is 120000 milliseconds or two minutes.
4. Select **Allow access on timeout** if you want the servlet to gain access to the session and continue normal execution even if the session is still locked by another servlet. If you do not select this box, the servlet execution will abort when the session request times out.
5. Click **Apply**.
6. Click **Save**.

Cookie settings

Use this page to configure cookie settings for session management.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management** → **Enable cookies**.

Cookie name

Specifies a unique name for the session management cookie. The servlet specification requires the name JSESSIONID. However, for flexibility, you can configure this value.

Restrict cookies to HTTPS sessions

Specifies that the session cookies include the secure field. Enabling this feature restricts the exchange of cookies to HTTPS sessions only.

Cookie domain

Specifies the domain field of a session tracking cookie. This value controls whether or not a browser sends a cookie to particular servers. For example, if you specify a particular domain, session cookies are sent to hosts in that domain. The default domain is the server.

Cookie path
Specifies that a cookie is sent to the URL designated in the path. Specify any string representing a path on the server. ″/″ indicates root directory. Specify a value to restrict the paths to which the cookie is sent. By restricting paths, you prevent the cookie from going to certain URLs on the server. If you specify the root directory, the cookie is sent no matter which path on the given server is accessed.

Cookie maximum age
Specifies the amount of time that the cookie lives on the client browser. Specify that the cookie lives only as long as the current browser session, or to a maximum age. If you choose the maximum age option, specify the age in seconds. This value corresponds to the Time to Live (TTL) value described in the Cookie specification.

Default is the current browser session which is equivalent to setting the value to -1.

Session management custom properties
You can specify additional settings for session management through setting custom properties.

Session management properties, like the session management configuration, can be configured at the server, application, or Web module level. The following steps are for setting the custom properties for session management at the server level.

1. In the administrative console click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management**.
2. Under **Additional Properties** select **Custom Properties**.
3. On the Custom Properties page, click **New**.
4. On the settings page, enter the property that you want to configure in the **Name** field and the value that you want to set it to in the **Value** field.
5. Click **Apply** or **OK**.
6. Click **Save** on the console task bar to save your configuration changes.
7. Restart the server.

CloneSeparatorChange
Use this property to maintain session affinity. The clone ID of the server is appended to session identifier separated by colon. On some Wireless Application Protocol (WAP) devices, a colon is not allowed. Set this property to ″true″ to change clone separator to a plus sign (+).

HttpSessionCloneId
Use this property to change the clone ID of the cluster member. Within a cluster, this name must be unique to maintain session affinity. When set, this name overwrites the default name generated by WebSphere Application Server.

Default clone ID length: 40

HttpSessionIdLength
Use this property to configure the session identifier length. Do not use an extremely low value; using a low value results in reduced number of combinations possible, thereby increasing risk of guessing the session identifier. In a cluster, all cluster members should be configured with same ID length. Allowed range: 8 to 128. Default length: 23.

HttpSessionReaperPollInterval
Use this property to set a wake-up interval for the process that removes invalid sessions. Setting this property overrides the default installation value, which is between 30 and 360 seconds. If the maximum inactive interval is less than 2 minutes, the reaper poll interval may be as short as 30 seconds. If the maximum inactive interval is more than 15 minutes, the reaper poll interval can be as long as 6 minutes. Because the default timeout and maximum inactive interval is 30 minutes, the reaper interval is usually

between 5 and 6 minutes. Set this property if you want to ensure that the reaper process runs at a specific interval. Use this property when you want the installation timed out sessions invalidated more frequently than 5 to 6 minutes. For example, setting HttpSessionReaperPollInterval=120 ensures that sessions are invalidated within 2 minutes of timing out. The minimum value for this property is 30 seconds. If a value less than the minimum is entered, the specified property is ignored and an appropriate value is automatically determined and used. The maximum inactive interval is the session timeout. The default is based on maximum inactive interval set in session management.

Data type	Integer
Units	Seconds

NoAdditionalSessionInfo
Set this value to ″true″ to force removal of information that is not needed in session identifiers.

SessionIdentifierMaxLength
Use this value to set maximum length that a session identifier can grow.

This property helps to find out the condition and take appropriate action to address servers fail-over. When this is specified, message is logged when specified maximum length is reached. Allowed value: integer.

SessionRewriteIdentifier
Use this property to change the key used with URL rewriting. Default key: jsessionid.

Servlet21SessionCompatibility
Set this custom property to true to enable global session behavior. In Servlet 2.2 and later, sessions are scoped at the Web module level. The default is false.

Note: This property is deprecated. The IBMApplicationSession method replaces the function of the Servlet21SessionCompatibility custom property.

SessionTableName
Use this custom property to set the database table name. Allowed value: String. The default value is SESSIONS.

Some applications may rely on method ejbCreate(...) to have created the entity bean in the database. For such a requirement, setting the JVM property *com.ibm.websphere.ejbcontainer.allowEarlyInsert* to **true** overrides the default behavior.

UseInvalidatedId
Set this custom property to true to reuse the incoming ID if the session with that ID was recently invalidated. This is a performance optimization because it prevents checking the persistent store. The default value is true.

UseOracleBLOB
The *UseOracleBLOB* custom property creates the HTTP session database table using the Binary Large Object (BLOB) data type for the medium column. This property increases performance of persistent sessions when Oracle databases are used. Due to an Oracle restriction, BLOB support requires use of the Oracle's oci database driver for more than 4000 bytes of data. You must also ensure that a new sessions table is created before the server is restarted by dropping your old sessions table or by changing the datasource definition to reference a database that does not contain a sessions table.

To create a sessions table using the BLOB data type, use the following name-vaule pair:

Name	Value
UseOracleBLOB	true

DebugSessionCrossover

The *DebugSessionCrossover* custom property enables code to perform additional checks to verify that only the session associated with the request is accessed or referenced. Messages are logged if any discrepancies are detected.

To enable session data crossover detection, use the following name-vaule pair:

Name	Value
DebugSessionCrossover	true

See article, "HTTP session problems" on page 24, for additional information.

HttpSessionIdReuse

The custom property *HttpSessionIdReuse* determines whether the session manager can use the session ID sent from a browser to preserve session data across Web applications that are running in an environment that is not configured for session persistence.

In a multi-JVM environment that is not configured for session persistence setting this property to `true` enables the session manager to use the same session information for all of a user's requests even if the Web applications that are handling these requests are governed by different JVMs. The default value for this property is `false`. To enable the session manager ton use the session ID sent from a browser to preserve session data across Web applications that are running in an environment that is not configured for session persistence, use the following name-value pair:

Name	Value
HttpSessionIdReuse	true

OptimizeCacheIdIncrements

Set the *OptimizeCacheIdIncrements* custom property to true to make the session manager assess whether the in-memory session is older than the copy in persistent store. Setting this property resolves the continually increasing cache ID.

If HTTP session management is configured to use session persistence and the user's browser session is moving back and forth across multiple Web applications you might see extra persistent store activity as the in-memory sessions are refreshed from the persistent store. As a result, the cache IDs are continually increasing and the in-memory session attributes are overwritten by those of the persistent copy. To prevent the cache IDs from continually increasing, use the following name-value pair:

Name	Value
OptimizeCacheIdIncrements	true

If the configuration is a cluster, ensure that the system times of each cluster member is identical as possible.

AlwaysEncodeURL

The Servlet 2.5 specification specifies to not encode the URL on a response.encodeURL call if it is not necessary. To support backward compatibility when URL encoding is enabled, set the *AlwaysEncodeURL* custom property to `true` to call the encodeURL method. The URL is always encoded, even if the browser supports cookies. Use the following name-value pair to encode all URLs:

Name	Value
AlwaysEncodeURL	true

UsingApplicationSessionsAndInvalidateAll

When the invalidateAllSet method is called, not all IBMApplicationSessions objects are checked. If you are using both the IBMApplicationSessions object and the invalidateAll call, use the following name-value pair:

Name	Value
UsingApplicationSessionsAndInvalidateAll	true

ForceSessionInvalidationMultiple

The *ForceSessionInvalidationMultiple* custom property indicates whether the session manager should wait indefinitely for a request to complete before attempting to invalidate the session, or should attempt to invalidate a session after the specified time limit has elapsed. The default value for this property is 1.

- If you specify 0 (zero) for this custom property, the session manager waits indefinitely until a request is complete before attempting to invalidate the session.

  If your requests normally are not bound by a response time limit, pecify 0 for this property.
- If you specify a positive integer, such as 1, 2, or 3, for this custom property, even if a session is not known to have completed, the session manager attempts to invalidate the session, if the indicated time period since the last access occurred has elapsed. This time period is the result of multiplying the value specified for this property and the value specified for the Session Timeout property. For example, if you specify 2 minutes for the Session Timeout property and 2 for the ForceSessionInvalidationMultiple property, the session manager attempts to invalidate the session after 4 minutes.

  If you want to invalidate your sessions after a certain amount of time has elapsed, specify the appropriate positive integer for this property.

Name	Value
ForceSessionInvalidationMultiple	1

Configuring session tracking for Wireless Application Protocol (WAP) devices

Applications that run in a Web container use sessions to keep track of individual users. Because most Wireless Application Protocol (WAP) devices do not support cookies, you can configure WAP devices to use URL rewriting to track sessions.

About this task

On most WAP devices, the maximum URL length is 128 characters. With URL rewriting, a session identifier is added to the URL itself, effectively decreasing the space available for the actual URL and the number of parameters that can be sent on a request.

To reduce the length of session identifier, you can configure key (jsessionid), session ID length and clone ID. To make these configuration changes, complete the following steps.

1. Open the administrative console.
2. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Web Container Settings** → **Web container**.
3. Under Additional Properties, click **Custom properties**.
4. Add the appropriate properties from the following list:
   - HttpSessionIdLength
   - SessionRewriteIdentifier
   - HttpSessionCloneId
   - CloneSeparatorChange
   - NoAdditionalSessionInfo

- SessionIdentifierMaxLength

5. Click **Apply** and **Save**.

Configuring for database session persistence

You can configure a database to collect session data for database session persistence.

About this task

To configure the session management facility for database session persistence, complete the following steps.

1. Create and configure a JDBC provider.
2. Create a data source pointing to a database.

   Use the JDBC provider that you defined: **Resources > JDBC > JDBC Providers >** *JDBC_provider* **> Data Sources > New**. The data source should be non-JTA, for example, non-XA enabled. Note the JNDI name of the data source.

   Point to an existing database.
3. Verify that the correct database is listed under **Resources > JDBC Providers >** *JDBC_provider* **> Data Sources >** *datasource_name*. If necessary, contact your database administrator to verify the correct database name.
4. Go to the appropriate level of Session Management.
5. Under Additional Properties, click **Distributed Environment Settings**
6. Select and click **Database**.
7. Specify the Data Source JNDI name from a previous step. The database user ID and password are case-sensitive.
8. Specify the database user ID and password that is used to access the database and for table creation. When you created your data source, you might have specified a Container Managed Authentication Alias or a Component Managed Authentication Alias; however, these two settings are not used by the session manager for session persistence. The session manager uses the userId and password specified in this step for session persistence.
9. Optional: Append the schema name in the session User ID field if you want to have more than one instance of the session table. The session manager uses the schema name to qualify the session table name for all database operations. If only the userid is specified without the schema name, the schema name defaults to NULL and therefore a table name with NULL as the schema name, for example, NULL.SESSIONS, is created. You can pass the create multiple session tables with different schema names, other than NULL, and access them separately by modifying the user name to contain the appropriate schema name. Use the following format to pass the schema name: userid::schemaName.
10. Retype the password for confirmation.
11. Configure a table space and page sizes for DB2 session databases.
12. Switch to a multirow schema.
13. Click **OK**.
14. If you want to change the tuning parameters, click **Custom Tuning Parameters** under Additional properties and select a setting or customize a setting.
15. Click **Apply**.
16. Click **Save**.

Switching to a multirow schema

The multirow schema configuration supports storing an unlimited amount of data that is only bounded by the database capacities in an application. The application can read individual fields instead of the whole record, which can help to improve performance by avoiding unnecessary Java object serialization.

Configure the session management facility to store each attribute in a session object in its own row in the database by using the multirow schema configuration.

About this task

The only practical limit that remains is the size of the session attribute object. The multirow schema potentially has performance benefits in certain usage scenarios, such as when larger amounts of data are stored in the session but only small amounts are specifically accessed during a given servlet processing of an HTTP request. In such a scenario, avoiding unneeded Java object serialization is beneficial to performance.

In addition to allowing larger session records, using multirow schema can yield performance benefits. However, it requires a little work to switch from single-row to multirow schema, as shown in the following table.

By default, a single session maps to a single row in the database table used to hold sessions. With this setup, there are hard limits to the amount of user-defined, application-specific data that WebSphere Application Server can access.

Consider configuring direct single-row usage to one database and multirow usage to another database while you verify which option suits your application needs. Do this in code by switching the data source used; then monitor performance.

Programming issue	**Application scenario**
Reasons to use single-row	• You can read or write all values with just one record read and write. • This takes up less space in a database because you are guaranteed that each session is only one record long.
Reasons **not** to use single-row	2-megabyte limit of stored data per session.
Reasons to use multirow	• The application can store an unlimited amount of data; that is, you are limited only by the size of the database and a 2-megabyte-per-record limit. • The application can read individual fields instead of the whole record. When large amounts of data are stored in the session but only small amounts are specifically accessed during servlet processing of an HTTP request, multirow sessions can improve performance by avoiding unneeded Java object serialization.
Reasons **not** to use multirow	If data is small in size, you probably do not want the extra overhead of multiple row reads when you can store everything in one row.

In the case of multirow usage, design your application data objects not to have references to each other, to prevent circular references. For example, suppose you are storing two objects A and B in the session using HttpSession.put(..) method, and A contains a reference to B. In the multirow case, because objects are stored in different rows of the database, when objects A and B are retrieved later, the object graph between A and B is different than stored. A and B behave as independent objects.

1. Modify the Session Management facility properties to switch from single to multirow schema.
2. Manually drop the table.

   To drop the table:
   a. Determine which data source configuration Session Management is using.
   b. In the data source configuration, look up the database name.
   c. Use the database facilities to connect to the database.

d. Drop the SESSIONS table.

Configuring tablespace and page sizes for DB2 session databases

If you are using DB2 for session persistence, you can increase the page size to optimize performance for writing large amounts of data to the database. Page sizes of 8K, 16K, and 32K are supported.

About this task

To use a page size other than the default (4K), complete the following steps.

1. If the SESSIONS table already exists, drop it from the DB2 database.
2. Create a new DB2 buffer pool and table space, specifying the same page size (8K, 16K or 32K) for both, and assign the new buffer pool to this table space.

```
DB2 Connect to session
DB2 CREATE BUFFERPOOL sessionBP SIZE 1000 PAGESIZE 8K
DB2 Connect reset
DB2 Connect to session
DB2 CREATE TABLESPACE sessionTS PAGESIZE 8K MANAGED BY SYSTEM
    USING ('D:\DB2\NODE0000\SQL00005\sessionTS.0') BUFFERPOOL sessionBP
DB2 Connect reset
```

   Refer to DB2 product documentation for details.
3. Configure the correct table space name and page size in the Session Management facility. Page size is referred to as *row size* on the Session Management page.)

Results

When the product is restarted, the Session Management facility creates the new SESSIONS table in the specified tablespace based on the indicated page size.

Creating a table for session persistence

You can use a database table to collect and store session data. If you are using a database table for session persistence, you must create and define a database table that is associated with the application server.

About this task

Whenever the session manager is set for database persistence, the session manager creates a table for its use. If you want to expand the column size limits to make it more appropriate for your Web site, you can create the table externally. If the external table is specified as the target table in the session manger database persistence configuration, then during the session manager start up, the external table is used. In most cases it is better to let the session manager create the table during startup.

To create a table for collecting session data, do the following:

1. Have your administrator create a database table for storing your session data using one of the following data definition language (DDL): For DB2:

```
CREATE TABLE <SchemaName>.sessions  (
  ID               VARCHAR(128) NOT NULL ,
  PROPID           VARCHAR(128) NOT NULL ,
  APPNAME          VARCHAR(128) NOT NULL,
  LISTENERCNT      SMALLINT ,
  LASTACCESS       BIGINT,
  CREATIONTIME     BIGINT,
  MAXINACTIVETIME  INTEGER ,
  USERNAME         VARCHAR(256) ,
```

```
  SMALL            VARCHAR(3122)  FOR BIT DATA ,
  MEDIUM           LONG VARCHAR FOR BIT DATA ,
  LARGE            BLOB(2M)
    )
```

For Oracle:

```
CREATE TABLE SESSIONS  (
  ID               VARCHAR(128) NOT NULL ,
  PROPID           VARCHAR(128) NOT NULL ,
  APPNAME          VARCHAR(128) NOT NULL,
  LISTENERCNT      SMALLINT ,
  LASTACCESS       INTEGER,
  CREATIONTIME     INTEGER,
  MAXINACTIVETIME  INTEGER ,
  USERNAME         VARCHAR(256) ,
  SMALL            RAW(2000),
  MEDIUM           LONG RAW ,
  LARGE            RAW(1)
  )
```

If the Web container custom property UseOracleBLOB is set to true then:

```
CREATE TABLE SESSIONS  (
  ID               VARCHAR(128) NOT NULL ,
  PROPID           VARCHAR(128) NOT NULL ,
  APPNAME          VARCHAR(128) NOT NULL,
  LISTENERCNT      SMALLINT ,
  LASTACCESS       INTEGER,
  CREATIONTIME     INTEGER,
  MAXINACTIVETIME  INTEGER ,
  USERNAME         VARCHAR(256) ,
  SMALL            RAW(2000),
  MEDIUM           BLOB,
  LARGE            RAW(1)
  )
```

Note:

a. At run time, the session manager accesses the target table using the identity of the J2EE server in which the owning Web application is deployed. Any Web container that is configured to use persistent sessions must have both read and update access to the subject database table.

b. HTTP session processing uses the index defined using the `CREATE INDEX` statement to avoid database deadlocks. In some situations, such as when a relatively small table size is defined for the database, DB2 may decide not to use this index. When the index is not used, database deadlocks can occur. If database deadlocks occur, see the DB2 Administration Guide for the version of DB2 you are using for recommendations on how to calculate the space required for an index and adjust the size of the tables that you are using accordingly.

c. It might be necessary to tune DB2 to make efficient use of the sessions database table and to avoid deadlocks when accessing it. Your DB2 administrator should refer to the DB2 Administration Guide for specific information about tuning the version of DB2 that you are using.

2. .

Database settings

Use this page to specify the settings for database session support.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Session management** → **Distributed environment settings** → **Database**.

Datasource JNDI name
Specifies the datasource description.

The JNDI name of the non-XA enabled datasource from which session management obtains database connections. For example, if the JNDI name of the datasource is ″jdbc/sessions″, specify ″jdbc/sessions.″ The datasource represents a pool of database connections and a configuration for that pool (such as the pool size). The datasource must already exist as a configured resource in the environment.

User ID
Specifies the user ID for database access.

Password
Specifies the password for database access.

DB2 row size
Specifies the table space page size configured for the sessions table, if using a DB2 database. Possible values are 4, 8, 16, and 32 kilobytes (KB). The default row size is 4KB.

The default row size is 4KB. In DB2, it can be updated to a larger value. This can help database performance in some environments. When this value is other than 4, you must specify table space name to use this property. For 4KB pages, the table space name is optional.

Table space name
Specifies that table space to be used for the sessions table.

This value is required when the DB2 page size is other than 4KB.

Use multi row schema
Specifies that each session data attribute is placed in a separate row in the database, allowing larger amounts of data to be stored for each session. This action can yield better performance when session attributes are very large and few changes are required to the session attributes. If use multi row schema is not enabled, instances of application data can be placed in the same row.

Configuring Web module class loaders

You can set values that control the class-loading behavior of an installed Web module.

Before you begin

This topic assumes that you installed a Web module on an application server.

About this task

Configure the class loader order value of an installed Web module. By default, a Web module has its own Web application archive (WAR) class loader to load the contents of the Web module, which are in the WEB-INF/classes and WEB-INF/lib directories.

An application class loader is the parent of a WAR class loader. The WAR class-loader policy value of an application class loader determines whether the WAR class loader or the application class loader is used to load the contents of the Web module.

The default WAR class loader policy value is `Class loader for each WAR file in application`. If the policy is set to `Class loader for each WAR file in application`, then each Web module receives its own class loader whose parent is the application class loader. If the policy is set to `Single class loader for application` on the settings page of an application class loader, then the application class loader loads the Web module contents as well as the enterprise bean (EJB) modules, shared libraries, resource adapter

archives (RAR files), and dependency Java archive (JAR) files associated to an application. Thus, the configuration of the parent application class loader affects the WAR class loader.

Use the administrative console to configure the application and WAR class loaders.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

1. If you have not done so already, configure the application class loader.

   Settings such as **Override class reloading settings for Web and EJB modules**, **Polling interval for updated files** and **WAR class loader policy** can affect Web module class loading.

   If **WAR class loader policy** is set to `Class loader for each WAR file in application`, then the Web module receives its own class loader and the WAR class-loader policy of the Web module defines the mode for a WAR class loader. If the policy is set to `Single class loader for application`, then the application class loader loads the Web module contents.

2. Specify the class loader order for the installed Web module.

   The Web module class-loader mode specifies whether the class loader searches in the parent application class loader or in the WAR class loader first to load a class. The default is to search in the parent application class loader before searching in the WAR class loader to load a class.

   Select either of the following values for **Class loader order**:

Option	Description
Classes loaded with parent class loader first	Causes the class loader to search in the parent application class loader first to load a class. This is the standard for Development Kit class loaders and WebSphere Application Server class loaders. **Note:** If classes and resources needed by the Web module cannot be accessed by the application class loader, but can be accessed by the WAR class loader, specify `Classes loaded with local class loader first (parent last)`. If the application class loader cannot find a class, the class loader delegates the request to find the class to its parent, the WebSphere Application Server extensions class loader. If the WebSphere Application Server extensions class loader cannot find the class, the class loader delegates the request to its parent, the bootstrap, extensions, and CLASSPATH class loaders created by the Java virtual machine. Requests can only go to a parent class loader; they cannot go to a child class loader. Thus, if `Classes loaded with parent class loader first` is specified, the WAR class loader never receives a request to load a class.
Classes loaded with local class loader first (parent last)	Causes the class loader to search in the WAR class loader first to load a class. By specifying `Classes loaded with local class loader first (parent last)`, your WAR class loader can override classes contained in the parent application class loader. **Note:** Specifying the `Classes loaded with local class loader first (parent last)` value might result in LinkageErrors or ClassCastException messages if you have mixed use of overridden classes and non-overridden classes.

3. Click **OK**.

What to do next

Save the changes to the administrative configuration.

Backing up and recovering servlets

Servlet source and class files, user profile data, Hypertext Transfer Protocol (HTTP) configuration, and administrative configuration should be considered for backup when using servlets. You should consider saving your HTTP configuration because changes to the HTTP configuration are often made to enable WebSphere Application Server to serve servlets and JSP requests, and to enable WebSphere Application Server security. You should consider backing up the user proifle data if you use the User Profile function of WebSphere Application Server.

- Backup servlet source and class files. Application code and configuration such as bindings, is located by default in the *profile_root*/installedApps directory. By saving this directory, you save your installed applications, including HTML, servlets, JavaServer Pages (JSP) files, and enterprise beans. Normally, each application is located in a separate subdirectory, so you can choose to save all applications or a subset.
  1. Save all installed applications. The commands below have been wrapped for display purposes. Enter each as a single command.

     ```
     SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
      OBJ(('/profile_root/installedApps'))
     ```

  2. Saves the sampleApp application only. The commands below have been wrapped for display purposes. Enter each as a single command.

     ```
     SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
      OBJ(('/profile_root/installedApps/cellName/sampleApp.ear'))
     ```

  If you have located utility or general purpose classes in other directories, such as *profile_root*/lib/app or *profile_root*/lib/ext, be sure to include those locations in your backup plan as well.
- Save your HTTP configuration.

  Note: The following information applies to IBM HTTP Server for iSeries (powered by Apache). If you are using Lotus® Domino® HTTP Server, see the Notes.net Documentation Library.

  1. Save the HTTP server instances for IBM HTTP Server for iSeries (powered by Apache). The HTTP server instances for IBM HTTP Server for iSeries (powered by Apache) are members of the QATMHINSTC file in the library QUSRSYS. An example save command for this file could be the following: SAVOBJ OBJ(QATMHINSTC) LIB(QUSRSYS) DEV(*SAVF) OBJTYPE(*FILE) SAVF(WSALIB/WSASAVF)
  2. Save the HTTP configurations for IBM HTTP Server for iSeries (powered by Apache). The HTTP configurations for IBM HTTP Server for iSeries (powered by Apache) are stored in the integrated file system in a subdirectory, chosen when the configuration was created. The recommended location is within the WebSphere instance directory. You can determine this file location by inspecting HTTP server instance member in the QATMHINSTC file in library QUSRSYS. An example save command for this file could be the following: SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file') OBJ(('*profile_root*/profile/apache/conf') ('*profile_root*/profile/htdocs')) where profile is the name of your instance. The default instance name is default.

Backing up and recovering JavaServer Pages files

JavaServer Pages source and generated servlet classes, Hypertext Transfer Protocol (HTTP) configuration, and administrative configuration should be considered for backup when using JavaServer Pages files.

- Save installed applications. Application code and configuration such as bindings, is located by default in the *profile_root*/installedApps directory. By saving this directory, you save your installed applications, including HTML, servlets, JavaServer Pages (JSP) files, and enterprise beans. Normally, each application is located in a separate subdirectory, so you can choose to save all applications or a subset.

1. Save all installed applications. The command below has been wrapped for display purposes. Enter the following as a single command, with a space between the end of DEV parameter and OBJ.

   ```
   SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
      OBJ(('/profile_root/installedApps'))
   ```

2. Save the sampleApp application only. The command below has been wrapped for display purposes. Enter the following as a single command with a space between the end of DEV parameter and OBJ.

   ```
   SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
      OBJ(('/profile_root/installedApps/cellName/sampleApp.ear'))
   ```

- Save and restore your JSP files. When JSP files are run, a servlet class is generated, compiled, and then run. When saving and restoring your JSP files, you can elect to save only the JSP source or the generated files as well.
  - If you save and restore only the JSP source, the servlet source and class files are regenerated when they are invoked. This is a simpler, smaller save and restore operation. Note that regeneration slows the first requests, and default optimization is done on the generated Java programs.
  - If you save and restore the source and generated files, no regeneration is done. If you have optimized Java programs to levels other than the default, this optimization is preserved.

Example

WebSphere Application Server places the generated files (*.class, .java*, and optionally, *.dat*) in a temporary directory under the WebSphere Application Server instance. For example, the default instance stores the generated files in this directory:

/profile_root/temp/*node_name*/*application_server*/*enterprise_app*/*web_module*

In this example:

- *profile* is the name of your instance. The default instance name is default.
- *node_name* is the name of the iSeries server or partition on which your WebSphere Application Server instance is running
- *application_server* is the name of your WebSphere Application Server
- *enterprise_app* is the name of the enterprise application to which the JSP file belongs
- *web_module* is the Web module that contains your JSP file.

Note: A .dat file is a helper file used by the generated servlet.

Chapter 2. Portlet applications

Task overview: Managing portlets

You can use this task to manage deployed portlet applications.

Before you begin

Before you begin this task, you must have a portlet application installed. See Installing enterprise application files for additional information.

About this task

You can complete the following steps to manage portlets.

- Render a portlet.
  - Access a single portlet using “Portlet Uniform Resource Locator (URL) addressability” on page 106.
  - Access multiple portlets using “Portlet aggregation using JavaServer Pages” on page 100.
- Change the location of “Portlet preferences” on page 108. By default, portlet preferences for each portlet window are stored in a cookie. However, you can change the location of where to store portlet preferences.
- Disable URL addressability. By default, you can access a portlet through an Uniform Resource Locator (URL), however, you can disable this feature.
- Enable portlet fragment caching. Portlet fragment caching is disabled by default.

Portlets

Portlets are reusable Web modules that provide access to Web-based content, applications, and other resources. Portlets can run on WebSphere Application Server because it has an embedded JSR 286 Portlet container. The JSR 286 API provides backwards compatibility. You can assemble portlets into a larger portal page, with multiple instances of the same portlet displaying different data for each user.

From a user’s perspective, a portlet is a window on a portal site that provides a specific service or information, for example, a calendar or news feed. From an application development perspective, portlets are pluggable Web modules that are designed to run inside a portlet container of any portal framework. You can either create your own portlets or select portlets from a catalog of third-party portlets.

Each portlet on the page is responsible for providing its output in the form of markup fragments to be integrated into the portal page. The portal is responsible for providing the markup surrounding each portlet. In HTML, for example, the portal can provide markup that gives each portlet a title bar with minimize, maximize, help, and edit icons.

You can also include portlets as fragments into servlets or JavaServer Pages files. This provides better communication between portlets and the Java Platform, Enterprise Edition (Java EE) Web technologies provided by the application server.

If you use Rational Application Developer version 6 (RAD) to create your portlets, you must remove the following reference to the std-portlet.tld from the web.xml file to run the portlets outside of RAD:

```
<taglib id="PortletTLD">
 <taglib-uri>http://java.sun.com/portlet</taglib-uri>
 <taglib-location>/WEB-INF/tld/std-portlet.tld</taglib-location>
</taglib>
```

Also if you use RAD version 6 to create portlets, note that portlets created by using the Struts Portlet Framework are not supported on WebSphere Application Server.

Portlet applications

If the portlet application is a valid Web application written to the Java Portlet API, the portlet application can operate on both the Portal Server and the WebSphere Application Server without requiring any changes. JSR 168 and JSR 286 compliant portlet applications must not use extended services provided by WebSphere Portal to operate on the WebSphere Application Server.

Portlet container

The *portlet container* is the runtime environment for portlets using the JSR 286 Portlet specification, in which portlets are instantiated, used, and finally destroyed. The JSR 286 Portlet API provides standard interfaces for portlets and backwards compatibility for JSR 168 portlets. Portlets based on this JSR 286 Portlet Specification are referred to as standard portlets.

A simple portal framework is provided by the PortletServingServlet servlet. The PortletServingServlet servlet registers itself for each Web application that contains portlets. You can use the PortletServingServlet servlet to directly render a portlet into a full browser page by a URL request and invoke each portlet by its context root and name. See “Portlet Uniform Resource Locator (URL) addressability” on page 106 for additional information. If you want to aggregate multiple portlets on the page, you need to use the aggregation tag library. See the article “Portlet aggregation using JavaServer Pages” for additional information. The PortletServingServlet servlet can be disabled in an extended portlet deployment descriptor called the ibm-portlet-ext.xmi file.

Remote request dispatcher support for portlets

The remote request dispatcher (RRD) support allows the invocation of portlets outside of the current Java virtual machine (JVM) within an Network Deployment single core group environment. The request related data is passed to the remote JVM where the portlet is invoked. The response is transmitted back and processed on the local JVM. Thus it guarantees that URLs contained in the portlet markup are created according to the local portal context. The remote request dispatcher support is only provided for JSR 168 compliant portlets.

Portlet container settings

Use this page to configure and manage the portlet container of this application server.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Portlet Container Settings** → **Portlet container**.

Enable portlet fragment cache

Specifies whether to create a cached entry when a portlet is invoked, similar to servlet caching of the Web container settings.

Portlet fragment caching requires that servlet caching is enabled. Therefore, enabling portlet fragment caching automatically enables servlet caching. Disabling servlet caching automatically disables portlet fragment caching.

Portlet aggregation using JavaServer Pages

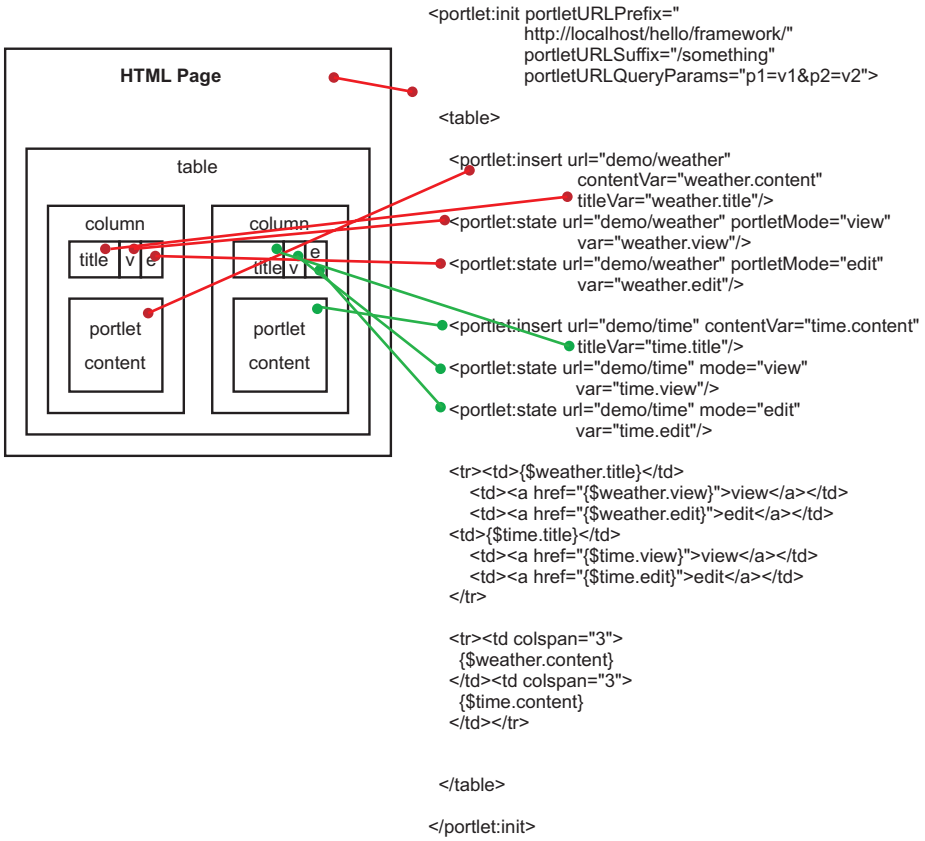
The aggregation tag library generates a portlet aggregation framework to address one or more portlets on one page If you write JavaServer Pages, you can aggregate multiple portlets on one page using the aggregation tag library. This tag library does not provide full featured portal aggregation implementation, but provides a good migration scenario if you already have aggregating servlets and JavaServer Pages and want to switch to portlets.

To allow the customer to create a simple portal aggregation, the aggregation tag library also provides the following features.

- Invoke a portlet's action method
- Render multiple portlets on one page
- Provide links to change the portlet's mode or window state
- Display the portlet's title
- Retain the portlet cookie state

The aggregation tag library and JavaServer Pages that use the aggregation tag library will only work with the WebSphere Application Server portlet container implementation because the protocol between the tags and the container is not standardized.

The following diagram depicts how an HTML page would look like and what tags are used in order to create the page. See "Aggregation tag library attributes" on page 102 for information on the aggregation tag library attributes.



When you use the aggregation tag library, you must set the portletUrlPrefix attribute of the init tag to the aggregating application. You need to:

- Ensure that the portletUrlPrefix attribute is set to the following in the aggregator page.

  `"http://" + <server_address> + ":" + <server_port> + "/" + <aggregator context> + "/" <aggregator mapping>`
- Reference the aggregation JSP page within the web.xml file through a servlet mapping ending with `/*`. For example, `/aggregation/*`

When aggregating multiple portlets on a single page, special care must be used with the naming conventions of form attribute names in your portlets. Because your portlets are all on the same page, they all share the same HttpServletRequest. When one portlet is viewed the entire page is refreshed and form data is re-posted. Therefore, if there are multiple portlets that are aggregated on a single page with the same form attribute names, there could be logic corruption when form data is re-posted.

Aggregation tag library attributes

The aggregation tag library is used to aggregate multiple portlets on one page. This topic describes the attributes within the aggregation tag library.

Supported arguments include:

init

This tag initializes the portlet framework and has to be used in the beginning of the JSP. All other tags described in this section are only valid in the body of this tag, therefore the init tag usually encloses the whole body of a JSP. In case the current URL contains an action flag the action method of the corresponding portlet is called. The state and insert tags are sub-tags of the init tag.

The init tag has the following attributes:

- portletURLPrefix = ”<any string>”

  This URL defines the prefix used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet’s render method, which is called by using the insert tag. This is a required attribute.
- portletURLSuffix = ”<any string>”

  This URL defines the suffix used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet’s render method, which is called by using the insert tag. This is attribute optional.
- portletURLQueryParams = ”<any string>”

  This URL defines the query parameters used for PortletURLs. Portlet URLs are created either by the state tag or within a portlet’s render method, which is called by using the insert tag. This is attribute optional.

scope, portlet

The scope tag and portlet tag are used to provide information that is necessary when a portlet application is installed under a multiple part context root, for example, /context1/context2. These tags add a render parameter to the newly created URL.

The urlParam tag has the following attributes:

- context = ″/<context1>/<context2>″

  Specifies the context root of the portlet application in which the portlet is deployed. This attribute is required.
- portletname = ″<portlet-name>″

  Specifies the portlet-name. This attribute is required.
- windowId = ”<any string>”

  Defines the window ID for the concrete portlet instance. This attribute is required.

The following is an example of how to use the scope and portlet tags:

```
<%@ taglib uri="http://ibm.com/portlet/aggregation" prefix="portlet" %>

<portlet:scope>
        <portlet:portlet context="/myportletcontext1/myportletcontext2" portletname="MyPortlet" windowId="sample"/>
</portlet:scope>

<portlet:init portletURLPrefix="/myportalcontext/ ">
....
</portlet:init>
```

state

The state tag creates a URL pointing to the given portlet using the given state. You can place this URL either into a variable specified by the var attribute or you can write it directly to the output stream. This tag is useful to create URLs for HTML buttons, images, and other items such that when the URL is invoked, the state changes defined in the URL are applied to the given portlet.

The state tag has the following attributes:

- url = ”<context>/<portlet-name>”

  Identifies the portlet for this tag by using the context and portlet-name to address the portlet. This attribute is required.
- windowId = ”<any string>”

  Defines the window ID for the portlet URL created by this tag. This is attribute optional.
- var = ”<any string>”

  If defined the URL is written into a variable with the given scope and name, not to the output stream. This is attribute optional.
- scope = ”page|request|session|application”

  This attribute is only valid if the var attribute is specified. If defined, the URL is not written to the output stream but a variable is created in the given scope with the given name. The default is page. This is attribute optional.
- portletMode = ”view|help|edit|<custom>”

  This attribute sets the portlet mode.
- portletWindowState = ”maximized|minimized|normal|<custom>”

  This attribute sets the window state.
- action = ”true/false”

  This attribute defines whether this is an action URL. This is attribute optional. The default is false.

urlParam

Adds a render parameter to the newly created URL.

The urlParam tag has the following attributes:

- name = ”<any string>”

  Indicates the name of the parameter. This is attribute required.
- value = ”<any string>”

  Indicates the value of the parameter. This is attribute required.

insert

This tag calls the render method of the portlet and retrieves the content as well as the title. You can optionally place the content and title of the specified portlet into variables using the contentVar and titleVar attributes.

The insert tag has the following attributes:

- url = ”<context>/<portlet-name>” (mandatory) Identifies the portlet for this tag by using the context and portlet-name to address the portlet

  This is attribute required.
- windowId = ”<any string>”

  Defines the window ID of the portlet. This is attribute optional.
- contentVar = ”<any string>”

  If defined, the portlet’s content is not written to the output stream but written into a variable with the given scope and name. This is attribute optional.
- contentScope = ”page|request|session|application”

  This attribute is only valid if the contentVar tag is used. If defined, the portlet’s content is written into a variable with the given scope and name, not to the output stream. The default is page. This is attribute optional.
- titleVar = ”<any string>”

  If defined the portlet’s title is written into a variable with the given scope and name. If it is not defined, the title is ignored and not written to the output stream. This is attribute optional.
- titleScope = ”page|request|session|application”

This attribute is only valid if titleVar tag is used. If defined, the portlet's title is written into a variable with the given scope and name, not to the output stream. The default is page. This is attribute optional.

Example: Using the portlet aggregation tag library

You can use the aggregation tag library to aggregate multiple portlets to have multiple and different content on one page. The library can be used by every JavaServer Pages (JSP) file that has been included by a servlet.

To use the portlet aggregation tag library, you must declare the tag-lib at the top of the JSP file using, `<%@ taglib uri="http://ibm.com/portlet/aggregation" prefix="portlet" %>`, as in the following example. The following JSP file example shows how to aggregate portlets on one page.

```
<%@ taglib uri="http://ibm.com/portlet/aggregation" prefix="portlet" %>
<%@ page isELIgnored="false" import="java.util.Enumeration"%>

<portlet:init portletURLPrefix="/dummy/portletTagTest/" portletURLSuffix="/more" portletURLQueryParams="p1=v1&p2=v2">


   <portlet:insert url="worldclock/StdWorldClock" contentVar="worldclockcontent" titleVar="worldclocktitle"/>
    <portlet:state url="worldclock/StdWorldClock" portletMode="view" var="worldclockview"
       portletWindowState="maximized">
        <portlet:urlParam name="namea" value="valuea"/>
        <portlet:urlParam name="nameb" value="valueb"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" portletMode="edit" var="worldclockedit" portletWindowState="normal">
        <portlet:urlParam name="name1" value="value1"/>
        <portlet:urlParam name="name2" value="value2"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" portletMode="view" var="worldclockmin"
       portletWindowState="minimized">
        <portlet:urlParam name="namemin" value="valuemin"/>
        <portlet:urlParam name="namemin" value="valuemin"/>
    </portlet:state>

    <portlet:insert url="worldclock/StdWorldClock" windowId="min" contentVar="simplecontent" titleVar="simpletitle"/>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="view" var="simpleview"
  portletWindowState="maximized">
        <portlet:urlParam name="name3" value="value3"/>
        <portlet:urlParam name="name4" value="value4"/>
        <portlet:urlParam name="name5" value="value5"/>
        <portlet:urlParam name="name5" value="value5a"/>
        <portlet:urlParam name="name5" value="value5b"/>
        <portlet:urlParam name="name5" value="value5c"/>
    </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="edit" var="simpleedit"
  action="true" portletWindowState="normal">
        <portlet:urlParam name="name6" value="value6"/>
        <portlet:urlParam name="name6" value="value6z"/>
            </portlet:state>
    <portlet:state url="worldclock/StdWorldClock" windowId="min" portletMode="view" var="simplemin"
  portletWindowState="minimized">
        <portlet:urlParam name="name1" value="value1"/>
        <portlet:urlParam name="name2" value="value2"/>
    </portlet:state>

    <portlet:insert url="test/TestPortlet1" contentVar="testcontent" titleVar="testtitle"/>
    <portlet:state url="test/TestPortlet1" portletMode="view" var="testview" portletWindowState="maximized"/>
    <portlet:state url="test/TestPortlet1" portletMode="edit" var="testedit" portletWindowState="maximized"/>


<!--  This table is the outtermost table for creating two-column portal layout  -->
<TABLE border="0" CELLPADDING="3" CELLSPACING="8" WIDTH="100%">
<TR>
<TD VALIGN="top">
```

```
<!--  This table is the top portlet in the first column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet top left">
    <tr><td class="portletTitle" NOWRAP>worldclock title:${worldclocktitle}</td>
        <td CLASS="portletTitleControls"  NOWRAP>
            <a href="${worldclockview}">view</a>
            <a href="${worldclockedit}">edit</a>
            <a href="${worldclockmin}">minimize</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${worldclockcontent}
     </td>
    </tr>
  </table>

  <BR/>

<!--  This table is the bottom portlet in the first column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet bottom left">
    <tr>
    <td class="portletTitle" NOWRAP>test title:${testtitle}</td>
        <td CLASS="portletTitleControls" NOWRAP>
            <a href="${testview}">view</a>
            <a href="${testedit}">edit</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${testcontent}
     </td>
    </tr>
  </table>


  </TD>



  <TD VALIGN="top">


<!--  This table is the top portlet in the second column -->

  <table border="0" width="100%" CELLPADDING="3" CELLSPACING="0" CLASS="portletTable" SUMMARY="portlet top right">
  <tr>
        <td class="portletTitle" NOWRAP>simple title:${simpletitle}</td>
        <td CLASS="portletTitleControls" NOWRAP>
            <a href="${simpleview}">view</a>
            <a href="${simpleedit}">edit</a>
            <a href="${simplemin}">minimize</a>
        </td>
    </tr>
    <tr>
    <td CLASS="portletBody" COLSPAN="2">
      ${simplecontent}
     </td>
    </tr>
  </table>

    </TD>
    </TR>
```

```
    </table>


</portlet:init>
```

You can include the following formatting to the previous example JSP file immediately after declaring the tag library.

```
<STYLE TYPE="TEXT/CSS">
BODY {
    font-family:Verdana,sans-serif; font-size:70%
}
.portletTitle {
    text-align: left;border-top: #000000 1px solid; border-bottom: #000000 1px solid; FONT-SIZE: 60.0%;
    COLOR: #ffffff; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif; BACKGROUND-COLOR: #5495d5;
}
.portletTitleControls {
    text-align: right;border-top: #000000 1px solid; border-right: #000000 1px solid; border-bottom: #000000
    1px solid; FONT-SIZE: 60.0%; COLOR: #ffffff; FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif;
    BACKGROUND-COLOR: #5495d5;
}
.portletTitleControls A {
    COLOR: #ffffff; text-decoration:none; border:#5495d5 1px solid;border-left:white 1px solid;
    padding-left:0.5em; padding-right:0.5em;
}
.portletTitleControls A:hover {
    COLOR: #ffffff; text-decoration:none; border-top:white 1px solid;
   border-bottom:white 1px solid;border-right:white 1px solid;
}
.minimizeControl {
    font-weight:bold; font-size:100%;
}
.portletTable {
    border-left: gray 1px solid;
    border-bottom: gray 1px solid;
    border-right: gray 1px solid;
}
.portletBody {
    font-family:Verdana,sans-serif; font-size:70%
}

</STYLE>
```

Portlet Uniform Resource Locator (URL) addressability

You can request a portlet directly through a Uniform Resource Locator (URL) to display its content without portal aggregation. The PortletServingServlet servlet registers each Web application that contains portlets. It is similar to the FileServingServlet servlet of the Web container that serves resources. The PortletServingServlet servlet supports direct rendering of portlets into a full browser page by a URL request.

You can invoke each portlet by its context root and name with the URL mapping /<portlet-name> that is created for each portlet. The context root and name has the following format:

```
http://<host>:<port>/<context-root>/<portlet-name> For example,
 http://localhost:9080/portlets/TestPortlet1
```

The context root identifies the Web archive (WAR) file that contains the portlet. The portlet name uniquely identifies the portlet with a portlet application of a WAR file. The DefaultDocumentFilter servlet only supports HTML, UTF8 encoding and an extended URL form based on the basic URL form as shown above.

You can only display one portlet at a time using the PortletServingServlet servlet. If you want to aggregate multiple portlets on the page, you need to use the aggregation tag library. See the article “Portlet aggregation using JavaServer Pages” on page 100 for additional information.

Because a portlet only delivers fragment output whereas a servlet usually delivers document output, a mechanism is introduced to convert the fragment into a document, called the PortletDocumentFilter filter. By default, the PortletDocumentFilter filter only supports converting HTML. The conversion is implemented using a servlet filter before the PortletServingServlet servlet is initiated to return the portlet's content inside of a document. This default document servlet filter only applies to URL requests, not for includes or forwards using the RequestDispatcher method. You can create servlet filters to support other markups additional document servlet filters. See the article, Converting portlet fragments to an HTML document, for additional information.

The PortletServingServlet servlet does not persist portlet preferences in a XML file or database. It places the portlet preferences directly into a cookie to store the preferences persistently. See the article, "Portlet preferences" on page 108, for additional information on how to change this behavior.

Portlet URL syntax

You can add additional portal context such as portlet mode or window state. You can access the PortletServingServlet servlet by using a URL mapping that has the following structure:

```
http://host:port/context/portlet-name [/portletwindow[/ver [/action |
 /resource[/id=custom-id][/cacheability]] [/mode] [/state] [rparam][/?name]]]
```

Any differing URL structure results in a com.ibm.wsspi.portletcontainer.InvalidURLException exception. Empty strings are not permitted as parameter values and creates an InvalidURLException exception. The following is a list of valid parameters:

http:// host:port/context/*portlet-name*
: This is the minimum URL required to access a portlet. A default portlet window called 'default' is created. The *portlet-name* variable is case-sensitive.

/portletwindow
: This parameter identifies the portlet window. You must set this parameter if you choose to add more portal context information to the URL.

/ver=*major.minor*
: This optional parameter is used to define the version of the portlet API that is used. You must set this parameter if you choose to add more portal context information to the URL. Only versions '1.0' and '2.0' are supported. Any other version creates an InvalidURLException exception.

/action
: This is a required parameter if you call the action method of the portlet. The action parameter causes the action process of the portlet to be called. After the action has been completed, a redirect is automatically issued to call the render process. To control the subsequent render process, a document servlet filter can set a request attribute with name 'com.ibm.websphere.portlet.action' and value 'redirect' to specify that the portlet serving servlet directly returns after action without calling the render process.

/mode=view | edit | help | *custom-mode*
: This optional parameter defines the portlet mode that is used to render the portlet. The default mode is 'view'. The value is not case-sensitive. For example, 'View', 'view' or 'VIEW' results in the same mode.

/state=normal | maximized | minimized | *custom-state*
: This optional parameter defines the window state that is used to render the portlet. The default state is 'normal'. The value is not case-sensitive, for example, 'Normal', 'normal', or 'NORMAL' results in the same state.

*** [/rparam=***name* ***[=***value***]]**
: This optional parameter specifies render parameters for the portlet. Repeat this parameter chain to provide more than one render parameter. For example, /rparam=invitation/ rparam=days=Monday=Tuesday.

?name=*value***&***name2***=***value2* **...**
Query parameters may follow optionally. They are not explicitly supported by the portlet container, but they do not invalidate the URL format.

/action | /resource
This parameter defines the methods of the portlet that is called. Valid values are no, action or resource parameter. No specific method defined calls the render method. The resource parameter is only supported for JSR 286 portlets.

/resource [/id=custom-id] [/cacheability=cacheLevelFull | cacheLevelPortlet | cacheLevelPage]
Set this parameter to define the method of the portlet to be called. No redirection occurs. No other method of the portlet is called. To control the resource parameter, you can add an additional ID parameter to provide a resource serving identifier that is passed through to the portlet. The cacheability parameter defines the cache level of this resource URL. This parameter is only supported with JSR 286 portlets .

The following list includes examples of valid JSR 168 and JSR 286 URLs:

- http:// your.server.name:9080/sample/WorldClock
- http:// your.server.name:9080/sample/WorldClock/myPortlet/ver=1.0/mode=edit/rparam=timezone=UTC
- http:// your.server.name:9080/sample/WorldClock/myPortlet/ver=1.0/action/state=maximized?timezone=UTC
- http://your.server.name:9080/sample/WorldClock/myPortlet/ver=2.0/resource/id=somePicture.jpg

Portlet preferences

Preferences are set by portlets to store customized information. By default, the PortletServingServlet servlet stores the portlet preferences for each portlet window in a cookie. However, you can change the location to store them in either a session, an .xml file, or a database.

Storing portlet preferences in cookies

The attributes of the cookie are defined as follows:

Path
context/*portlet-name*/*portletwindow*

Name:
The name of the cookie has the fixed value of **PortletPreferenceCookie**.

Value
The value of the cookie contains a list of preferences by mapping to the following structure:

```
*['/' pref-name *['=' pref-value]]
```

All preferences start with '/' followed by the name of the preference. If the preference has one or more values, the values follow the name separated by the '=' character. A null value is represented by the string '#*!0_NULL_0!*#'. As an example, the cookie value may look like, /locations=raleigh=boeblingen/regions=nc=bw

Customizing the portlet preferences storage

You can override how the cookie is handled to store preferences in a session, an .xml file or database. To customize the storage, you must create a filter, servlet or JavaServer Pages file as new entry point that wraps the request and response before calling the portlet. Examine the following example wrappers to understand how to change the behavior of the PortletServingServlet to store the preferences in a session instead of cookies.

The following is an example of how the main servlet manages the portlet invocation.

```
public class DispatchServlet extends HttpServlet
{
    ...
    public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
 response.setContentType("text/html");

 // create wrappers to change preference storage
 RequestProxy req = new RequestProxy(request);
 ResponseProxy resp = new ResponseProxy(request, response);

 // create url prefix to always return to this servlet
                ...
                req.setAttribute("com.ibm.wsspi.portlet.url.prefix", urlPrefix);

 // prepare portlet url
 String portletPath = request.getPathInfo();
                ...

 // include portlet using wrappers
 RequestDispatcher rd = getServletContext().getRequestDispatcher(modifiedPortletPath);
                rd.include(req, resp);
    }
}
```

In the following example, the request wrapper changes the cookie handling to retrieve the preferences out of the session.

```
public class RequestWrapper extends HttpServletRequestWrapper
{
    ...
    public Cookie[] getCookies() {
        Cookie[] cookies = (Cookie[]) session.getAttribute("SessionPreferences");
        return cookies;
    }
}
```

In the following example, the response wrapper changes the cookie handling to store the preferences in the session:

```
public class ResponseProxy extends HttpServletResponseWrapper
{
   ...
    public void addCookie(Cookie cookie) {
        Cookie[] oldCookies = (Cookie[]) session.getAttribute("SessionPreferences");
        int newPos = (oldCookies == null) ? 0 : oldCookies.length;
        Cookie[] newCookies = new Cookie[newPos+1];
        session.setAttribute("SessionPreferences", newCookies);

        if (oldCookies != null) {
        System.arraycopy(oldCookies, 0, newCookies, 0, oldCookies.length);
        }
        newCookies[newPos] = cookie;
    }
}
```

Example: Configuring the extended portlet deployment descriptor to disable PortletServingServlet

Portlet URL serving supports direct access to all functions and states of a portlet by creating the appropriate URLs. In a production setup where the portlet is served through an enterprise portal application that applies its own access control, is considered a security risk. By setting the portletServingEnabled property to false, an administrator can ensure that a sensitive portlet is never accessed by direct URL serving.

Extensions for the portlet deployment descriptor are defined within a file called ibm-portlet-ext.xmi. This deployment descriptor is an optional descriptor that you can use to configure WebSphere extensions for the portlet application and its portlets. For example, you can disable the PortletServingServlet servlet for the portlet application in the extended portlet deployment descriptor.

The ibm-portlet-ext.xmi extension file is loaded during application startup. If there are no extension files specified with this setting, the default values of the portlet container are used.

The default for the portletServingEnabled attribute is true. The following is an example of how to configure the application so that a PortletServingServlet servlet is not created for any portlet on the portlet application.

```
<?xml version="1.0" encoding="UTF-8"?>
<portletappext:PortletApplicationExtension xmi:version="1.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:portletappext="portletapplicationext.xmi"
    xmlns:portletapplication="portletapplication.xmi"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmi:id="PortletApp_ID_Ext"
    portletServingEnabled="false">
  <portletappext:portletApplication href="WEB-INF/portlet.xml#myPortletApp"/>
</portletappext:PortletApplicationExtension>
```

Portlet container custom properties

You can configure name-value pairs of data, where the name is a property key and the value is a string value that you can use to set internal system configuration properties. Defining a new property enables you to configure a setting beyond that which is available in the administrative console. The following is a list of the available Portlet container custom properties.

To specify Portlet container custom properties:

1. In the administrative console click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Portlet Container Settings** → **Portlet container**.
2. Under **Additional Properties** select **Custom Properties**.
3. On the Custom Properties page, click **New**.
4. On the settings page, enter the name of the custom property that you want to configure in the **Name** field and the value that you want to set it to in the **Value** field.
5. Click **Apply** or **OK**.
6. Click **Save** on the console task bar to save your configuration changes.
7. Restart the server.

Following is a list of custom properties provided with the Application Server.

Using short names for Portlet and PortletApplication MBeans

Portlet MBeans are registered by both their short name and full name. To enable the use of short MBean names, create the following name-value pair:

Name	Value
useShortMBeanNames	true

The default is false.

MBeans registered by the full identifiable name, have the following format:

```
<ApplicationName>#<WARfilename.war>_portlet.<portlet_name> for the Portlet MBean
<ApplicationName>#<WARfilename.war>_portlet for the PortletApplication MBean
```

where <..> is replaced by the corresponding application data. For example, `SampleApplication#SamplePortlet.war_portlet.SamplePortlet`.

Portlet and PortletApplication MBeans

The MBeans of type portlet and portletapplication provide information about a given portlet application and its portlets. Through the MBean of type portletapplication, you can retrieve a list of names of all portlets that belong to a portlet application. By querying the MBean of type portlet with a given portlet name, you can retrieve portlet specific information from the MBean of type portlet.

Each MBean that corresponds to a portlet or portlet application is uniquely identifiable by its name. Portlet applications are not required to have a name set within the portlet.xml. The MBean name for MBeans of the portletapplication type is the enterprise archive (EAR) file name followed by ″#″ and the Web module name concatenated with the string ″_portlet″. For example, portletapplication type MBeans have the following format:

```
<EarFileName>#<WarFileName>_portlet
```

The name chosen for the MBean of type portlet is the name of the MBean of type portletapplication that the portlet belongs to, concatenated with the portlet name:

```
<EarFileName>#<WarFileName>_portlet.<portletname>
```

The following is an example of the resulting PortletApplication MBean name and portlet names:

```
EarName            SampleEar
WebModule            SampleWar.war

PortletApplication MBean name:   SampleEar#SampleWar_portlet
Portlet:            SampleEar#SampleWar_portlet.BookmarkPortlet
```

The MBean names have been changed compared to version 6.1, because the old naming patterns are not unique and can lead to problems under certain circumstances. If you rely on the old naming pattern, you can set the portlet container custom property, `useShortMBeanNames`, to true to activate the previous known MBean names. Because this is a performance impact, you might not want to activate the old naming pattern if it is not necessary.

A full stop separates the preceding Web module name from the portlet name. Review the Portlet and PortletApplication MBean type API documentation for additional information. The generated API documentation is available in the information center table of contents from the path, **Reference** > **Administrator** > **API documentation** > **MBean interfaces**.

The following code is an example of how to invoke the MBean of type portletapplication for an application with the name, SampleWar.

```
String myPortletApplicationName = "SampleEar#SampleWar_portlet";
This name is composed by the Ear file name followed by "#" and
the Web module name concatenated with the substring "_portlet"


com.ibm.websphere.management.AdminService adminService =
  com.ibm.websphere.management.AdminServiceFactory.getAdminService();
javax.management.ObjectName on =
  new ObjectName("WebSphere:type=PortletApplication,name=" + myPortletApplicationName + ",*");

Iterator onIter = adminService.queryNames(on, null).iterator();
while(onIter.hasNext())
{
 on = (ObjectName)onIter.next();
}

String ctxRoot = (java.lang.String)adminService.getAttribute(on, "webApplicationContextRoot");
```

In the previous example, the MBeanServer is first queried for an MBean of type portletapplication. If this query is successful, the webApplicationContextRoot attribute is retrieved on that MBean or the first MBean that is found. The result is stored in the ctxRoot variable. This variable now contains the context root of the Web application that contains the portlet application that was searched. The variable is similar to ″/bookmark″.

The next code example demonstrates how to invoke the MBean of type portlet for a portlet with the name, BookmarkPortlet.

```
String myPortletName = "SampleEar#SampleWar_portlet.BookmarkPortlet";
This name is composed by the name of the MBean of type portletapplication and
the portlet name, separated by a full stop because the same portlet name may
be used within different Web modules, but must be unique within the system.


com.ibm.websphere.management.AdminService adminService =
  com.ibm.websphere.management.AdminServiceFactory.getAdminService();
javax.management.ObjectName on =
  new ObjectName("WebSphere:type=Portlet,name=" + myPortletName + ",*");
Iterator iter = adminService.queryNames(on, null).iterator();

while(iter.hasNext())
{
 on = (ObjectName)iter.next;
}

java.util.Locale locale = (java.util.Locale) adminService.getAttribute(on, "defaultLocale");
```

The locale returned by the method getAttribute method for the MBean is the default locale defined for this portlet.

Full names for Portlet and PortletApplication MBeans

MBeans are also registered by the full identifiable name:

```
<ApplicationName>#<WARfilename.war>_portlet.<portlet_name> for the Portlet MBean
<ApplicationName>#<WARfilename.war>_portlet for the PortletApplication MBean
```

where <..> is replaced by the corresponding application data. For example, `SampleApplication#SamplePortlet.war_portlet.SamplePortlet`. You can enable the short MBean names by setting the `useShortMBeanNames` portlet container custom property to true.

Portlet URL security

WebSphere Application Server enables direct access to portlet Uniform Resource Locators (URLs), just like servlets. This section describes security considerations when accessing portlets using URLs.

For security purposes, portlets are treated similar to servlets. Most portlet security uses the underlying servlet security mechanism. However, portlet security information resides in the `portlet.xml` file, while the servlet and JavaServer Pages files reside in the `web.xml` file. Also, when you make access decisions for portlets, the security information, if any, in the `web.xml` file is combined with the security information in the `portlet.xml` file.

Portlet security must support both programmatic security, that is `isUserInRole`, and declarative security. The programmatic security is exactly the same as for servlets. However, for portlets, the `isUserInRole` method uses the information from the security-role-ref element in `portlet.xml`. The other two methods used by programmatic security, `getRemoteUser` and `getUserPrincipal`, behave the same way as they do when accessing a servlet. Both of these methods return the authenticated user information accessing the portlet.

The declarative security aspect of the portlets is defined by the security-constraint information in the portlet.xml file. This is similar to the security-constraint information used for the servlets in the web.xml file with the following differences:

- The auth-constraint element, which lists the names of the roles that can access the resources, does not exist in the portlet.xml file. The portlet.xml file contains only the user-data-constraint element, which indicates what type of transport layer security (HTTP or HTTPS) is required to access the portlet.
- The security-constraint information in the portlet.xml file contains the portlet-collection element, while the web.xml file contains the web-resource-collection element. The portlet-collection element contains only a list of simple portlet names, while the web-resource-collection contains the url-patterns as well as the HTTP methods that need protection.

The portlet container does not deal with the user authentication directly. For example, it does not prompt you to collect the credential information. The portlet container must, instead, use the underlying servlet container for the user authentication mechanism. As a result, there is no auth-constraint element in the security-constraint information in the portlet.xml file.

In WebSphere Application Server, when a portlet is accessed using a URL, the user authentication is processed based on the security-constraint information for that portlet in the web.xml file. This implies that to authenticate a user for a portlet, the web.xml file must contain the security-constraint information for that portlet with the relevant auth-constraints contained in it. If a corresponding auth-constraint for the portlet does not exist in the web.xml file, it indicates that the portlet is not required to have authentication. In this case, unauthenticated access is permitted just like a URL pattern for a servlet that does not contain any auth-constraints in the web.xml file. An auth-constraint for a portlet can be specified directly by using the portlet name in the url-pattern element, or indirectly by a url-pattern that implies the portlet.

Note: You cannot have a servlet or JSP with the same name as a portlet for WebSphere Application Server security to work with portlet.

The following examples demonstrate how the security-constraint information contained in the portlet.xml and web.xml files in a portlet application are used to make security decisions for portlets. The security-role-ref element, which is used for isUserInRole calls, is not discussed here because it is used the same way for servlets.

In the examples below (unless otherwise noted), there are four portlets (MyPortlet1, MyPortlet2, MyPortlet3, MyPortlet4) defined in portlet.xml. The portlets are secured by combining the information, if any, in the web.xml file when they are accessed directly through URLs.

All of the examples show the contents of the web.xml and portlet.xml files. Use the correct tools when creating these deployment descriptor files as you normally would when assembling a portlet application.

Example 1: The web.xml file does not contain any security-constraint data

In the following example, the security-constraint information is contained in portlet.xml:

```
<security-constraint>
   <display-name>Secure Portlets</display-name>
   <portlet-collection>
   <portlet-name>MyPortlet1</portlet-name>
   <portlet-name>MyPortlet3</portlet-name>
   </portlet-collection>
   <user-data-constraint>
   <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

In this example, when you access anything under MyPortlet1 and MyPortlet3, and these portlets are accessed using the unsecured HTTP protocol, you are redirected through the secure HTTPS protocol. The transport-guarantee is set to use secure connections. For MyPortlet2 and MyPortlet4, unsecured (HTTP)

access is permitted because the transport-guarantee is not set. There is no corresponding security-constraint information for all four portlets in the web.xml file. Therefore, all of the portlets can be accessed without any user authentication and role authorization. The only security involved in this instance is the transport-layer security using Secure Sockets Layer (SSL) for MyPortlet1 and MyPortlet3.

The following table lists the security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
/MyPortlet1/*	HTTPS	None	None
/MyPortlet2/*	None	None	None
/MyPortlet3/*	HTTPS	None	None
/MyPortlet4/*	None	None	None

Example 2: The web.xml file contains portlet specific security-constraint data

In the following example, the security-constraint information that corresponds to the portlet is contained in web.xml. The portlet.xml file is the same as that shown in the previous example.

```
<security-constraint id="SecurityConstraint_1">
    <web-resource-collection id="WebResourceCollection_1">
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/MyPortlet1/*</url-pattern>
      <url-pattern>/MyPortlet2/*</url-pattern>
      </web-resource-collection>
    <auth-constraint id="AuthConstraint_1">
      <role-name>Employee</role-name>
    </auth-constraint>
</security-constraint>
```

The security-constraint information contained in the web.xml file in this example indicates that the user authentication must be performed when accessing anything under the MyPortlet1 and MyPortlet2 portlets. When you attempt to access these portlets directly using URLs, and there is no authentication information available, you are prompted to enter their credentials. After you are authenticated, the authorization check is performed to see if you are listed in the Employee role. The user/group to role mapping is assigned during the portlet application deployment. In the web.xml file listed above, note the following:

- Because the web.xml file uses url-pattern, the portlet names have been modified slightly. MyPortlet1 is now /MyPortlet1/*, which indicates that everything under the MyPortlet1 URL is protected. This matches the information in the portlet.xml file because the security runtime code converts the portlet-name element in the portlet.xml file to url-pattern (for example, MyPortlet1 to /MyPortlet1/*), even for the transport-guarantee.
- The http-method element in the web.xml file is not used in the example because all HTTP methods must be protected.

The following table lists the new security constraints that are applicable to the individual portlets.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Employee)
MyPortlet2/*	None	Yes	Yes (Employee)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	None	None

Example 3: The web.xml file contains generic security-constraint data implying all portlets.

In the following example, the security-constraint information is contained in the web.xml file that corresponds to the portlet. The portlet.xml file is the same as that shown in the first example.

```
<security-constraint id="SecurityConstraint_1">
    <web-resource-collection id="WebResourceCollection_1">
      <web-resource-name>Protected Area</web-resource-name>
      <url-pattern>/*</url-pattern>
      </web-resource-collection>
    <auth-constraint id="AuthConstraint_1">
      <role-name>Manager</role-name>
    </auth-constraint>
    </security-constraint>
```

In this example, /* implies that all resources that do not contain their own explicit security-constraints should be protected by the Manager role as per the URL pattern matching rules. Because the portlet.xml file contains explicit security-constraint information for MyPortlet1 and MyPortlet3, these two portlets are not protected by the Manager role, only by the HTTPS transport. Because the portlet.xml file cannot contain the auth-constraint information, any portlets that contain security-contraints in it are rendered unprotected when an implying URL (/* for example) is listed in the web.xml file because of the URL matching rules.

In the case above, both MyPortlet1 and MyPortlet3 can be accessed without user authentication. However, because MyPortlet2 and MyPortlet4 do not have security-constraints in the portlet.xml file, the /* pattern is used to match these portlets and are protected by the Manager role, which requires user authentication.

The following table lists the new security constraints that are applicable to the individual portlets with this setup.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	None	None
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

If in the example above, if you must also protect a portlet contained in the portlet.xml file (for example, MyPortlet1), the web.xml file should contain an explicit security-constraint entry in addition to /* as shown in the following example:

```
<security-constraint id="SecurityConstraint_1">
         <web-resource-collection id="WebResourceCollection_1">
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/*</url-pattern>
            </web-resource-collection>
         <auth-constraint id="AuthConstraint_1">
            <role-name>Manager</role-name>
          </auth-constraint>
      </security-constraint>
      <security-constraint id="SecurityConstraint_2">
         <web-resource-collection id="WebResourceCollection_2">
            <web-resource-name>Protection for MyPortlet1</web-resource-name>
            <url-pattern>/MyPortlet1/*</url-pattern>
            </web-resource-collection>
         <auth-constraint id="AuthConstraint_1">
            <role-name>Manager</role-name>
          </auth-constraint>
      </security-constraint>
```

In this case, MyPortlet1 is protected by the Manager role and requires authentication. The data-constraint of CONFIDENTIAL is also applied to it because the information in the web.xml file and the portlet.xml file

are combined. Because MyPortlet3 is not explicitly listed in the `web.xml` file, it is still not protected by the Manager role and does not require user authentication.

The following table shows the effect of this change.

URL	Transport Protection	User Authentication	Role Based Authorization
MyPortlet1/*	HTTPS	Yes	Yes (Manager)
MyPortlet2/*	None	Yes	Yes (Manager)
MyPortlet3/*	HTTPS	None	None
MyPortlet4/*	None	Yes	Yes (Manager)

Developing console modules

Console modules are Web applications that are accessed from Integrated Solutions Console. Console modules provide the business logic and transaction processes that enable administration functions.

About this task

The following skills are essential for developing and testing console modules.

- Java Platform, Enterprise Edition (Java EE)
- XML
- Portlet development using the Java Portlet Specification (JSR 168)

The console also supports existing modules that have been developed using Struts and Tiles APIs. This is only for legacy support. New console modules are supported only if they are developed using standard portlet APIs.

- Review the console module samples. The sample console modules provide examples of portal application archives and how to use the APIs and other features.
- Set up the development environment. You need an integrated development environment (IDE) that supports Java development to quickly develop, test, and deploy your portlet applications.
- Develop your first console module. This topic is for developers new to console module development. It is assumed that you already have an IDE prepared. The topics in this section take you through the process of creating a simple console module. The console module in this exercise consists of a single portlet which is deployed to a single page as a member of the sample console modules provided by IBM. Be sure you have successfully deployed the sample console modules before starting.
- Adding advanced API features. The class files for console modules are developed using the portlet API of the Java Portlet Specification (JSR 168). Integrated Solutions Console includes additional APIs for launching pages, passing properties to other console modules, and launching Eclipse-based help. This topic provides information about the APIs available to console modules.

Overview of Integrated Solutions Console

Integrated Solutions Console offers several advantages for implementing administration functions, as described in this topic.

Integrated Solutions Console provides a single, common interface for system administration. It provides the main platform on which IBM and non-IBM products can build administrative user interfaces as individual plug-ins to a common console framework. Standardizing product administration functions to run on the Integrated Solutions Console platform gives them a more common look and feel and a more consistent behavior, thereby reducing the learning curve and adoption as new management components are introduced. Administrators can interact with multiple IBM and non-IBM products from a single browser-based console.

Consistency across administrative interfaces

Integrated Solutions Console provides a common appearance (for example, theme, layout and banner) and behavior (for example, navigation and authentication) to enable consistent user interaction for administering software products.

A standards-based architecture

Integrated Solutions Console provides a standards-based architecture for Web administration. Each Integrated Solutions Console module consists of one or more Web applications that have access to services within the Java Platform, Enterprise Edition (Java EE) environment provided by WebSphere Application Server. The help interface is implemented using the Eclipse open standard. Console modules are developed using the Java Portlet Specification.

Easy deployment of product administration consoles

The Integrated Solutions Console framework provides an XML-based interface for deploying console modules to a console installation. XML descriptors provide the information needed to deploy the portlet, resources, and setup the page layout and navigation in the console. A console module can be easily removed without impact to the remaining console modules.

Accelerated development of solutions

Using Integrated Solutions Console enables products to reduce the time required to develop solutions that require administration functions. The standards-based architecture, common framework, and support for existing investment help reduce the time required to implement solutions.

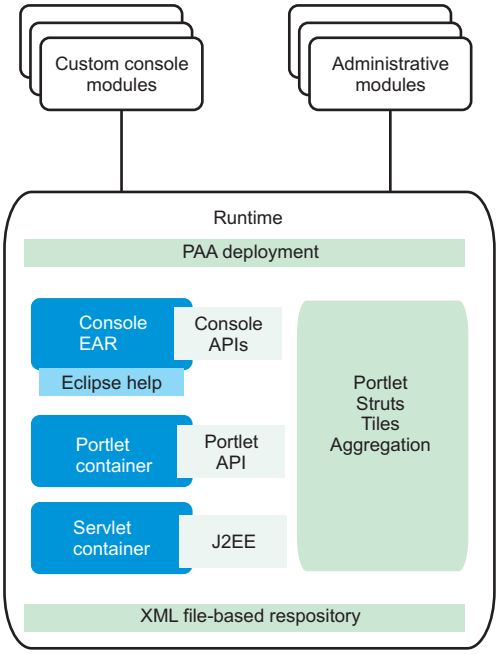
Improved administration efficiency

Customers invest significant resources in training administrative personnel. Providing a common interface across IBM products enables customers to reduce training time and expense.

Integrated Solutions Console components

This topic describes the components of the administrative console.

This figure illustrates the Integrated Solutions Console platform.



The following components and features support console modules running in Integrated Solutions Console.

- A common console

  Provides a common appearance and navigation behavior for console modules. For example, each portlet is rendered using a common skin, rendering the same title bar icons for actions such as editing preferences, maximizing, or requesting help for the portlet.
- Built-in administration console modules

  Provide security, log and trace, and other functions that apply to the entire the Integrated Solutions Console installation.
- A portlet container

  Supports portlets developed using the Java Portlet Specification. The current specification as of this writing is JSR 168.
- PAA deployment

  XML descriptors are provided for the developer to specify pages, navigation nodes, and access control for the console module, which are created at deployment. The descriptor can also register the following information about the console module.
  - Custom roles for the application in addition to the roles provided by the application server
  - An about page with links to useful information and resources for a product
  - A pointer to another console module that is the lead application for the product
  - Prerequisite applications and console modules that should be already deployed
  - Portlets that are already deployed on the server that should be reused for the console module
  - Multiple entities of a single portlet definition that can be used on different pages
  - External URLs that can be launched from the console navigation
  - Row and column layout of a page
  - Windows that contain portlets on the page
  - Empty windows on a page that are filled at runtime depending on the user's navigation selection
  - The order and nesting of navigation nodes used by the console module
  - Parent navigation nodes under which the navigation tree for the current console module are placed
  - Navigation nodes from other products that should be included under the current console module's navigation

- An application server

  Provides a complete Java runtime environment for console modules.
- APIs for console modules

  Supports the following actions:
  - Passing properties between portlets
  - Launching a page from a portlet
  - Invoking Eclipse-based help
- Eclipse Help

  Eclipse Help is provided as a Web application running on the same server as Integrated Solutions Console. Eclipse Help serves help documents for a single module as well as for the entire console.

What is new in Integrated Solutions Console

Learn about changes to the administrative console since the last release of WebSphere Application Server.

Customizing the banner

You can change the console banner to display custom text in the greeting. This could be useful, for example, in environments where you have multiple application servers and you want to make sure console users can determine which server they are administering. The same text is also displayed in the title bar of the browser window.

Administrative agent console

In environments that use the administrative agent to administer multiple application server nodes, you have a choice to select when logging in to the console. You can choose to log into the administrative agent or one of its registered profiles.

Example: Console module samples

The sample console modules provide examples of portal application archives and how to use the APIs and other features. These samples are available from the *Samples for WebSphere Application Server* page at the following location

http://www.ibm.com/developerworks/websphere/library/samples/index.html

These samples help understand the structure of a console module WAR package and the portal application descriptors that distinguish the application as a console module. After downloading the samples, follow the instructions in “Deploying a console module” on page 126. The following table lists the samples that are ready for deployment.

Sample	Description
Page layout (pagelayout.war)	Demonstrates how to specify the layout of console modules on a page using the elements of the portal topology descriptor. This sample shows how the lead application of a product provides an about page, which contains helpful information and resources about the product. The about page can be accessed from the console Welcome page. This module also demonstrates how to reuse a portlet from another module (Tree merge).
Tree merge (TreeMerge.war)	Demonstrates how a subordinate application deploys into the lead application for a product. The <PAA-ref/> element indicates the page layout sample as the lead application. Also, the <requires/> element indicates the page layout sample as a prerequisite that must already be deployed. The <parent-tree/> element merges the navigation nodes for the tree merge sample into the page layout navigation tree.

Sample	Description
Portlet context (PortletContext.war)	Demonstrates how to use the Dynamic UI Manager APIs to launch a page and how to pass properties to other console modules. The module receiving these properties can be on the same page or on another page.
Modes (Modes.war)	Demonstrates how to implement the Edit and Help modes in a module and how to invoke Eclipse-based help.
Basic module (Basicmodule.war)	Demonstrates how to implement basic module components.

Setting up the development environment

Set up your integrated development environment (IDE) for developing and debugging console modules.

You can use the Integrated Solutions Console Visual Designer feature to create the portal topology and security deployment descriptors needed to enable portlet applications to run in the administrative console environment. Visual Designer is available for download at http://www.ibm.com/support/docview.wss?rs=180&uid=swg24011666. This feature is supported using one of the following IDEs.

- IBM Application Server Toolkit, Version 6.1.1
- IBM Rational Application Developer, Version 7.0

After you download Visual Designer, you can install it to your development environment like any other feature. Refer to the documentation for your IDE for more information.

As you test your console modules, you might discover that changes that you make to the JSP files are not rendered when you log in to the application server. If so, you probably need to check the **reloadingEnabled** attribute in the extended deployment descriptor for the module (ibm-web-ext.xmi) Make sure this value is set to `true` to enable the JSP file to reload each time you deploy a new change. See the ″Hot deployment and dynamic reloading″ topic in the *Developing and deploying applications* PDF for more information.

Developing your first console module

Follow these steps if you are new to console module development.

Before you begin

You should have an integrated development environment (IDE) that supports Java programming setup. See “Setting up the development environment” for more information.

The topics in this section take you through the process of creating a simple console module. The sample console module consists of a single portlet which is deployed to a single page as part of the set of sample console modules provided by IBM. Be sure you have successfully deployed the sample console modules before starting.

- “Developing the portlet” on page 121
- “Creating the descriptors for the console module” on page 122
- “Packaging a console module” on page 125
- “Deploying a console module” on page 126
- “Testing a console module” on page 128
- “Removing a console module” on page 128

Developing the portlet

This topic is intended for developers who are unfamiliar with portlet development. A simple Java class and JSP are provided.

Developing the source Java class file

The following example shows the Java code for a portlet in its simplest form.

```
package com.ibm.isclite.samples.basicmodule;

import java.io.*;
import javax.portlet.*;

public class BasicModule extends GenericPortlet {

 public void doView(RenderRequest request, RenderResponse response)
             throws PortletException, IOException {
  // Set the MIME type for the render response
  response.setContentType("text/html");

        // Invoke the JSP to render
  PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/jsp/basicView.jsp");
  rd.include(request,response);
 }

}
```

The portlet code must extend the GenericPortlet class and output for the response in the doView() method. Portlets are rendered in different modes. The initial mode when a portlet is called to render is the view mode. The response output is provided by a JSP, which provides markup that can be aggregated into a larger HTML page. The package name in this example is consistent with the console module samples.

Developing the JSP

The following shows a portlet JSP for view mode in its simplest form.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1" session="false" buffer="none"%>
<%@ page import="javax.portlet.*" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portletAPI" %>

<portletAPI:defineObjects />

<%
  PortletPreferences prefs = renderRequest.getPreferences();
  String URL = prefs.getValue("website","");
%>

<p><a name="<portletAPI:namespace/>basicAnchor">Basic contents</a></p>

<p>
  <a href="<%=URL%>" target="_blank">
    <img src='<%=renderResponse.encodeURL(renderRequest.getContextPath() + "/images/logo.gif")%>'
         alt="logo" />
      </a>
</p>
<p><em>Company logo with link.</em></p>
```

The taglib directive specifies the portlet tags from the Java Portlet Specification. The following tags are used in this example.

namespace
: Uniquely qualifies named HTML tags or JavaScript functions to prevent clashes with other portlets on the page using the same name.

defineObjects
Makes the RenderRequest, RenderResponse, and PortletConfig objects available to the JSP. This tag is used in this example so that the image can be rendered using the encodeURL() method of the render response, and to allow the JSP to read preferences in the portlet descriptor.

Creating the descriptors for the console module

Before you begin

The following descriptors are required for the console module.

- Web application descriptor (web.xml). This descriptor is required and described by the Java Servlet Specification.
- Portlet application descriptor (portlet.xml). This descriptor is required and described by the Java Portlet Specification. Refer to the console module samples for examples of how to create the portlet.xml.
- Portal topology descriptor (ibm-portal-topology.xml). This descriptor is unique to console modules and defines the portlets and resources in the module, how it is laid out on the page, and how it is accessed from the navigation.
- Portal security descriptor (ibm-portal-security.xml). This descriptor is unique to console modules and defines roles and access to navigation nodes and portlets.

About this task

The topology and security descriptors are the main component of console module packages that distinguish them from other portal application WAR files. The console module samples include these descriptors and demonstrate their features. The sample descriptors provide inline comments that explain the elements and how they are used.

For your first console module, you should use the Visual Designer plugin to the Application Server Toolkit to develop the descriptors. If you prefer to create them manually to become familiar with the structure of the XML, the steps in this topic show how to create simple descriptors that can be used to deploy an application into the samples provided by IBM. After following these steps, you should refer to the samples and the reference topics for the console module schemas for more complete information.

Portal topology descriptor

About this task

To follow these steps, open a file with the name ibm-portal-topology.xml in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into ibm-portal-topology.xml. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-topology
    xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd"
    xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
ibm-portal-topology.xsd">

   <application-definition appID="com.ibm.isclite.samples.basicModule" version="6.1">


   </application-definition>
   <PAA-ref>com.ibm.isclite.samples.PageLayout</PAA-ref>

</ibm-portal-topology>
```

   - The elements in this step provide the top level content of the topology descriptor.

- Most of the content for this descriptor will be placed inside the <application-definition/> element.
- The value of the **appID** attribute is consistent with the naming convention used by the sample console modules. The **version** attribute matches the version used by the sample console modules.
- The <PAA-ref/> element specifies the **appID** of the page layout sample, which is the lead application to which this application will be deployed.
- The following convention is used for unique name for all of the elements in the descriptor.

  ```
  namespace + element_type + identifier
  ```

2. Create a title for your application. The title is the first element in the application definition element.

   ```
   <title>
         <base:nls-string lang="en">My basic module</base:nls-string>
   </title>
   ```

3. Place the component tree after the <requires/> element as shown.

   ```
   <component-tree uniqueName="com.ibm.isclite.samples.basicModule.appTree">
      <portlet-definition uniqueName="com.ibm.isclite.samples.basicModule.portletDefinition.A">
         <title>
            <base:nls-string lang="en">My basic module</base:nls-string>
         </title>
         <resource-link name="basicModule"
                        portletApplication="com.ibm.isclite.samples.basicmodule.BasicModule.01a"
                        type="static"/>
      </portlet-definition>
      <portlet-entity   uniqueName="com.ibm.isclite.samples.basicModule.portletEntity.A"
                        portletDefinitionRef="com.ibm.isclite.samples.basicModule.portletDefinition.A">
         <title>
            <base:nls-string lang="en">My basic module</base:nls-string>
         </title>
      </portlet-entity>
   </component-tree>
   ```

   - The **name** attribute of the <resource-link/> element must specify the <portlet-name/> from the portlet deployment descriptor.
   - The **portletApplication** attribute of the <resource-link/> element must specify the **id** attribute of the <portlet-app/> element from the portlet deployment descriptor.
   - The **portletDefinitionRef** attribute of the <portlet-entity/> element references the unique name of the <portlet-definition/>. Subsequent elements in this descriptor use the unique name of the <portlet-entity/> to invoke the portlet. Multiple portlet entities can point to the same portlet definition. For the purpose of this exercise, a single portlet definition is provided with a single portlet entity.
   - The portlet title that is rendered in this example is derived from the title in the portlet entity. If that value is not provided, the portlet title is rendered by the title provided by the <window/> element in the layout tree. If that value is also not provided, the title is obtained from the <portlet-name/> element in the portlet.xml for the portlet indicated by the <resource-link/>.

4. Place the layout tree after the component tree as shown.

   ```
   <layout-tree>
      <layout-element uniqueName="com.ibm.isclite.samples.basicModule.layoutElement.A">
         <title>
            <base:nls-string lang="en">My basic module</base:nls-string>
         </title>
         <simple-container orientation="row"
                           uniqueName="com.ibm.isclite.samples.basicModule.container.A">
            <window uniqueName="com.ibm.isclite.samples.basicModule.window.A">
               <title>
                  <base:nls-string lang="en">My basic module</base:nls-string>
               </title>
               <component-definition-ref>
                  com.ibm.isclite.samples.basicModule.portletEntity.A
               </component-definition-ref>
   ```

```
            </window>
         </simple-container>
      </layout-element>
   </layout-tree>
```

- Each layout element in the layout tree defines the layout of a page in the console.
- Each page can have nested containers, and each simple container can have multiple windows. Each simple container has either a row or column orientation. The nesting of these containers can create a complex table structure for the page layout.
- The <component-definition-ref/> element specifies the unique name of the portlet entity that provides the content of this window.

5. Create the navigation element after the layout tree as shown.

```
<navigation-element uniqueName="com.ibm.isclite.samples.navigationElement.basicModule"
                    layoutElementRef="com.ibm.isclite.samples.basicModule.layoutElement.A">
   <title>
      <base:nls-string lang="en">Basic Module Sample</base:nls-string>
   </title>
   <preference name="ProductFilter">
      <base:value>
         <base:string>com.ibm.isclite.samples.PageLayout</base:string>
      </base:value>
   </preference>
   <parent-tree parentTreeRef="com.ibm.isclite.samples.navigationElement.Samples"/>
</navigation-element>
```

- Each navigation element defines a node in the console navigation. If you nest the navigation elements, it creates a hierarchical tree structure that is reflected in the console navigation. For the purpose of this exercise, only a single navigation element is provided. The **layoutElementRef** attribute indicates the unique name of the layout element that is rendered on the page that is displayed when this element is selected by the console user.
- The content of the `ProductFilter` preference indicates the **appID** of the lead application. This is used to render this navigation element when the lead product is selected by the console user for filtering the navigation.
- The content of the **parentTreeRef** element indicates the unique name of the parent navigation element for this element.

The following figure shows how this navigation element is rendered with the page layout sample. In this figure, the user has selected `Integrated Solutions Console Sample` from the **View** selection list to filter the navigation.

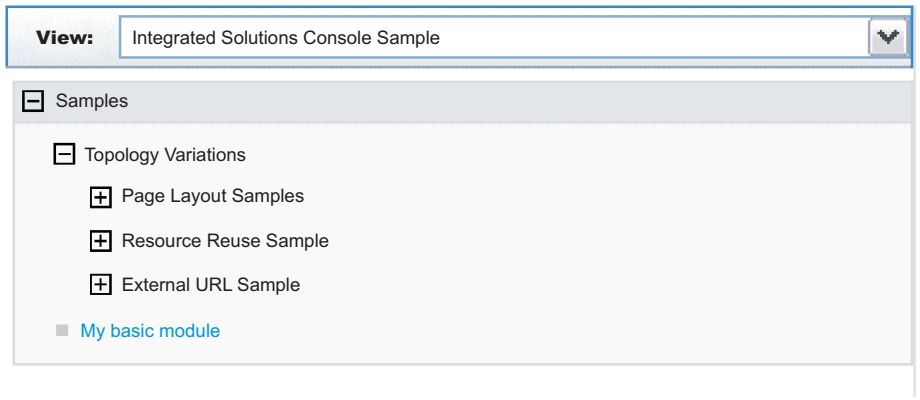


Figure 3. Sample console module navigation

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Portal security descriptor
Before you begin

To follow these steps, open a file with the name ibm-portal-security.xml in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into ibm-portal-security.xml. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-security
     xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd"
     xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
ibm-portal-topology.xsd
         http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd ibm-portal-base.xsd
         http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd ibm-portal-security.xsd">

  </ibm-portal-security>
```

2. Add the <application-role/> element as the content of the <ibm-portal-security/> element.

```
<application-role uniqueName="monitor">


</application-role>
```

   This sample identifies the administrative role, `monitor`, for access rights. See Portal security schema for a complete description of this element and the uniqueName attribute.

3. Add the following <portal-role/> elements as the content of the application role.

```
<portal-role object-ref="com.ibm.isclite.samples.navigationElement.basicModule"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.navigationElement.A"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.portletEntity.A"
             role-type="Privileged User"/>
```

   This sample assigns permission to a user with monitor access to view two navigation elements and to view, edit, and access help for one portlet. See Portal security schema for a complete description of this element and its attributes.

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Packaging a console module
Before you begin

Before you start this task, be sure your Java sources compiles without errors and the XML descriptors have been validated against the corresponding schemas.

About this task

An Integrated Solutions Console module must be packaged as a WAR file for deployment. For best results, use a comprehensive development tool such as Rational Application Developer. Otherwise, you can follow these steps to package your console module resources into the WAR file.

1. Create a working directory to collect all of the files that need to be packaged in the WAR file.
2. Organize your portlets and other resource files in the prescribed directory structure for portlet WAR files using the following steps:
   a. Place the following deployment descriptors in the directory /WEB-INF directory:
      - ibm-portal-topology.xml (topology descriptor)
      - ibm-portal-security.xml (security descriptor)
      - portlet.xml (portlet descriptor)
      - web.xml (Web application descriptor)
   b. Place individual Java classes for your portlets in the directory /WEB-INF/classes. Use a directory structure that reflects the fully-qualified class name of your portlet, for example, /com/ibm/isclite/samples/basicModule.
   c. Put JAR files in the directory /WEB-INF/lib.
   d. If the console module supports multiple languages, place the properties file that contains the translated strings (the resource bundles) in the directory /WEB-INF/classes/nls.
   e. Put any JavaServer Pages that must be served directly by the client in a directory that is not under the /WEB-INF directory (such /jsp). Resources in the /WEB-INF directory are protected and are not URL-addressable. JavaServer Pages that are accessed only by the portlets in the console module can be placed in the /WEB-INF path.
   f. If the console module has help files, copy the Eclipse help plug-in files to the directory /help/*plugin_name*, where *plugin_name* is the name of the Eclipse help plug-in, such as com.ibm.myprod.help.doc.
3. Use the Java JAR utility to create the WAR file. For instructions, see the Java Tutorial at http://java.sun.com/docs/books/tutorial/jar/basics/index.html . You can also use an assembly tool like Rational Application Developer to create the WAR file for the console module.

Deploying a console module

After you create the WAR file for a console module, you can deploy the console module to a system where the runtime for Integrated Solutions Console is installed and then test that the console module works correctly. Only one instance of a console module can be deployed on an Integrated Solutions Console installation. This rule applies even if the versions of the console module are different.

About this task

If you need to update the version of a console module, you must perform these steps.

1. Remove the console module.
2. Update the module with the required changes.
3. Reissue the deploy command.

To deploy a console module on a system where the runtime for Integrated Solutions Console is installed, perform the following procedure:

1. Go to the machine where the Integrated Solutions Console runtime is installed. You can also use a machine that has remote access to the drive where the runtime is installed.
2. Copy the console module WAR file to the *app_server_root*/systemApps/isclite.ear directory. As an alternative, you can extract the WAR file into a subdirectory of this location using the WAR file name for the subdirectory. For example, you would extract pagelayout.war to the *app_server_root*/systemApps/isclite.ear/pagelayout.war directory.
3. Grant the QEJBSVR profile the authority needed to complete the task. From an i5/OS command prompt, issue this command:

   ```
   CHGAUT OBJ('path_to_war') USER(QEJBSVR) DTAAUT(*RWX) OBJAUT(*ALL)
   ```

4. Start the wsadmin scripting client. If the server is not running, use the -conntype NONE option to run in local mode.

5. At the **wsadmin** command prompt, enter the command syntax. The following example presents the syntax:

```
$AdminApp update isclite modulefile {
-operation add
-contents path_to_war
-contenturi war_filename
-custom paavalidation=true
-custom containWebServices=true|false
-usedefaultbindings -contextroot /context
-MapWebModToVH {{.* .* admin_host}}
-MapModulesToServers {{.*  .*.war,.*  WebSphere:cell=cellname,node=nodename,server=servername}}}
```

Note:

- The values of the -contents and -contenturi options are case sensitive and should match the war file name that you are ready to deploy.
- Specifying `-custom paavalidation=true` causes the deployment to validate the XML descriptors for the console module package before deployment. This is optional.
- Set the option `-custom containWebServices=true` only if the package contains a Web service. This setting invokes more deployment processing to locate and setup the Web service. The default for this option is true. If the package does not contain any Web services, be sure to set this option to false for more efficient deployment.
- Use the -MapModulesToServers option when deploying to an agent profile.

For example:

```
$AdminApp update isclite modulefile {
-operation add
-contents app_server_root/systemApps/isclite.ear/pagelayout.war
-contenturi pagelayout.war
-custom paavalidation=true
-custom containWebServices=false
-usedefaultbindings  -contextroot /ibm/pagelayout
-contextroot /pagelayout
-MapWebModToVH {{.* .* admin_host}}
-MapModulesToServers {{.*  .*.war,.*  WebSphere:cell=cell_A,node=node_B,server=server_C}}}
```

The following messages are displayed if the deployment is successful.

```
Update of isclite has started.
ADMA5009I: An application archive is extracted at profile_root/wstemp/wstemp/app_10793d84b8e/ext/pagelayout.war
ADMA5064I: FileMergeTask completed successfully for isclite.
ADMA5005I: The application isclite is configured in the WebSphere Application Se
rver repository.
ADMA5005I: The application isclite is configured in the WebSphere Application Se
rver repository.
ADMA5110I: The application isclite is installed as a hidden application and will
 not be exposed via administrative interfaces such as GUI client, wsadmin or MBe
an Java API.  In order to perform management operations on this application, the
 application name must be known.
CWLAA10001I: The Integrated Solutions Console module was deployed successfully.
ADMA5005I: The application isclite is configured in the WebSphere Application Se
rver repository.
ADMA5011I: The cleanup of the temp directory for application isclite is complete
.
Update of isclite has ended.
```

6. After the deployment has completed successfully, save your work by issuing the following command:

```
$AdminConfig save
```

Results

If the deployment is successful, the WAR file contents are extracted to the /systemApps/isclite.war directory and the module metadata is copied to the following directory:

profile_root/config/cells/*cellname*/applications/isclite.ear/deployments/isclite

To view your console module, you must log back in to the console. Make sure the navigation is updated correctly. If so, select the new navigation nodes verify that the new pages and console modules are rendered correctly.

Testing a console module
About this task

Use the following checklist to test your console module after it has been deployed.

1. Log in to Integrated Solutions Console as an administrator.
2. If the module includes an <about-page/> element in the topology descriptor, verify that your suite and its version are listed in the table on the Welcome page. Also confirm that the product is listed in the **View** selection list for the navigation.
3. Verify that the console navigation tree contains all of the organizational nodes and pages that were defined. Verify that each node is at the correct level in the tree.
4. For each page, verify that the portlets on the page work correctly. If a portlet on the page launches a new page in work area, verify that the launching works correctly.
5. To test the access permissions specified in the console module descriptor, logout and then log in as a different user to verify that the correct pages are accessible.
6. If the console module included a help plug-in, make sure you can launch the help.
7. If the console module contains errors, refer to the SystemOut.log and SystemErr.log for any errors or messages. These logs are located in the following directory:

   profile_root/logs/server1

Removing a console module
About this task

Follow these steps to delete a console module.

1. Ensure that the server is started. You do not need to log in to the console.
2. Start the wsadmin scripting client.
3. Enter the following command from the wsadmin command prompt.

   ```
   $AdminApp update isclite modulefile {-operation delete -contenturi module_file_name.war}
   ```
4. Save your configuration changes.

   ```
   $AdminConfig save
   ```
5. Exit the wsadmin command environment.

   ```
   quit
   ```

Adding advanced API features

The class files for console modules are developed using the Java Portlet Specification. Integrated Solutions Console includes additional APIs for launching pages, passing properties to other portlets, and launching Eclipse-based help. This section provides information about the APIs available to console modules.

Developing portlets

Console modules are portlets that run in the portlet container of WebSphere Application Server.

Integrated Solutions Console supports console modules written to the Java Portlet Specification. A packaged console module WAR file contains one or more portlets or reuses portlets from other modules. According to the JSR 168 specification, a *portlet* is a Java technology based Web component, managed

by a portlet container, that processes requests and generates content that contributes to the overall portal page. Portlets share many of the same characteristics as servlets. However, portlets also exhibit differences. For example, portlets cannot set HTTP headers and can contribute only markup fragments to the portal page. The portlet specification leverages the functionality of the servlet specification as much as possible, and many parts of the Java Portlet Specification have been modeled after the servlet specification.

The application server includes a JSR 168-compliant portlet container to support deploying and running console modules in Integrated Solutions Console. The JSR 168 portlet container relies on the Java Platform, Enterprise Edition (Java EE) architecture implemented by the application server. Consequently, an Integrated Solutions Console module and its portlets are packaged and deployed similar to Java EE Web applications. In addition to the Web application descriptor and the portlet descriptor, the WAR file for an Integrated Solutions Console module includes additional descriptors needed to deploy the application to a console installation.

The following Web resources can useful for learning more about portlet development.

Resource	Web site URL
Portlet Specification 1.0	http://jcp.org/jsr/detail/168.jsp
Portlet Specification 2.0	http://jcp.org/en/jsr/detail?id=286
WebSphere Portal Zone	http://www.ibm.com/developerworks/websphere/zones/portal/
Best practices: Developing portlets using JSR 168 and WebSphere Portal	http://www.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html
WebSphere Portal external newsgroup	news://news.software.ibm.com/ibm.software.websphere.portal-server
Develop a Faces JSR 168 portlet using IBM Rational Application Developer 6.0 for autonomic computing	http://www.ibm.com/developerworks/edu/ac-dw-ac-jsfisc-i.html
Introduction to JSR 168 - The Portlet Specification	http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/index.html

Launching pages

Typically, users of the console launch pages using the links in the console navigation. However, portlets in a console module also launch pages directly using the Dynamic UI Manager API.

The DynamicUI Manager APIs are provided by 3 services: PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService. To use these services the portlet must first perform a JNDI lookup to get the PortletServiceHome object which you then can use to get the actual service interface object. This is demonstrated in the GetApps portlet of the PortletContext sample.. The following code is placed in a `try` block of the portlet's init() method.

Some information is split on multiple lines for printing purposes.

```
Context ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
// get propertyfactory service

propertyFactoryServiceHome = (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.
propertybroker.service.PropertyBrokerService");
if (propertyFactoryServiceHome == null)
   logger.log(Level.FINE, "no PropertyBrokerService propertyFactoryServiceHome");
else
   propertyFactoryService =

      (PropertyFactory)propertyFactoryServiceHome.getPortletService(com.ibm.portal.
propertybroker.service.PropertyBrokerService.class);
logger.log(Level.FINE, "propertyBrokerService="+propertyFactoryService);
```

```
// get dynamicUIManagerFactory service
dynamicUIManagerFactoryServiceHome =
   (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.DynamicUIManagerFactoryService");
if (dynamicUIManagerFactoryServiceHome == null)
   logger.log(Level.FINE, "no dynamicUIManagerFactoryService propertyFactoryServiceHome");
else
   dynamicUIManagerFactoryService =

      (DynamicUIManagerFactoryService)dynamicUIManagerFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.DynamicUIManagerFactoryService.class);
logger.log(Level.FINE, "dynamicUIManagerFactoryService="+dynamicUIManagerFactoryService);
// get urlGeneratorFactory service
urlGeneratorFactoryServiceHome =
   (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.URLGeneratorFactoryService");
if (urlGeneratorFactoryServiceHome == null)
   logger.log(Level.FINE, "no urlGeneratorFactoryService propertyFactoryServiceHome");
else
   urlGeneratorFactoryService =

      (URLGeneratorFactoryService) urlGeneratorFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.URLGeneratorFactoryService.class);
logger.log(Level.FINE, "urlGeneratorFactoryService="+urlGeneratorFactoryService);
```

A portlet can launch a new page using the dynamicUICtrl interface, passing the object ID of the page definition for the new page. The object ID is determined by performing a JNDI lookup on the unique name of the navigation element that launches the page. If you are planning to launch a navigation element, the navigation node must point to a layout element. In other words the navigation element must contain the l**ayoutElementRef** attribute. The addPage() method returns the object ID of the page instance, which should be saved by the portlet to create a URL and launch the page. The following excerpt from the PortletContext sample demonstrates how this works. The portlet GetApps.java includes a launchPage() method with this code.

```
Context ctx;
ObjectID pageDefinitionID = null;
String pagename="com.ibm.isclite.portletcontext.navigationElement.B";
logger.log(Level.FINE, "launch page=" + pagename);

PortletSession ps  = request.getPortletSession(false);
try {
    ctx = new InitialContext();
    pageDefinitionID =(ObjectID)ctx.lookup("portal:uniquename/"+pagename);
} catch (NamingException ne) {
    return;
}
try {
    PropertyController cproperty = propertyFactoryService.createProperty(myconfig);
    PropertyController cproperty2 = propertyFactoryService.createProperty(myconfig);

    cproperty.setType("String");
    cproperty.setName("servername");
    cproperty2.setType("String");
    cproperty2.setName("appDeployed");
    PropertyValue[] propertyValues = new PropertyValue[2];
    propertyValues[0] =
        propertyFactoryService.createPropertyValue(request, cproperty, serverName);
    propertyValues[1] =
        propertyFactoryService.createPropertyValue(request, cproperty2, appDeployed);

    DynamicUICtrl dmanagerCtrl =
       dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

    ObjectID newPageID =
        dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);

    RedirectURLGenerator urlGenerator =
        urlGeneratorFactoryService.getURLGenerator(request, response);
```

```
    EngineURL redirectURL = urlGenerator.createPageURL(newPageID);

    response.sendRedirect(redirectURL.toString());
} catch  ...
```

- The `pagename` String is created with the unique name of the navigation element (from the topology descriptor) that launches the page.
- The object ID is found by performing a JNDI lookup on this Context instance using this String:

  ```
  "portal:uniquename/"+pagename
  ```

  The String `portal:uniquename/` is required to obtain the object ID for the navigation element.
- Properties are created to be passed to the portlets on the page. The property names are `servername` and `appDeployed`.
- In addition to the request and response, you must include the String `isc.tasklaunch` as the configuration name on the getDynamicUICtrl() method.
- Use the addPage() method of the new DynamicUICtrl instance to create and instance of the page. Any properties that need to be passed to portlets on the new page must be provided in this method. The addPage() method does not launch the page, but it returns the object ID of the page instance. The next two methods, getURLGenerator() and createPageURL(), are used to construct the URL that opens the page. The URL contains property names and values if they have been provided.
- The URL to open the page is provided on the sendRedirect() method of the response.

Passing properties to other portlets

This topic describes how portlets can exchange properties. You should be familiar with the concepts for launching console pages before reading this topic.

Information can be provided to portlets on the same page using the addSharedPortlet() method. Property values are of the type com.ibm.portal.propertybroker.property.PropertyValue. The example in “Launching pages” on page 129 shows these properties passed to the page using an instance of the PropertyValue interface (`propertyValues`) and shows how that object was created using the PropertyFactory interface.

The SetServer.java source from the PortletContext sample shows how to pass properties to a portlet on the same page. It is assumed that the portlet has already performed a JNDI lookup to determine that the PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService services are available. This is described in “Launching pages” on page 129

- In this example, the portlet that is the source of the properties needs the object ID of the target portlet. The object ID is determined by performing a JNDI lookup using the values from the portlet’s <resource-link/> element in the topology descriptor.
- The property, `cproperty`, is created using the PropertyController interface and the createProperty() method of the PropertyFactory interface. The data type for the property must be a String.
- Prior to sending the property, the source portlet creates an instance of the DynamicUICtrl interface, passing the string ″isc.tasklaunch″ as the configuration name.
- The addSharedPortlet() method sends the properties to the target portlet on the request. After the action phase, the target portlet updates the output for the response.

```
Context ctx = null;
ObjectID portletDefinitionID1 = null;
ObjectID portletDefinitionID2 = null;

String portletname1="com.ibm.isclite.samples.PortletContext/PortletGetPerformance";
String portletname2="com.ibm.isclite.samples.PortletContext/PortletGetApps";

PortletSession ps  = request.getPortletSession(false);

try
   {
```

```
      ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
      portletDefinitionID1 =
         (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname1);
      portletDefinitionID2 =
         (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname2);
   } catch (NamingException ne)
     {
        logger.log(Level.FINE, "portletdefinitionID not found - Naming exception:"+ne.getMessage());
        return;
     }
logger.log(Level.FINE, "portletdefinitionID="+portletDefinitionID1.toString());

try
{
   PropertyController cproperty =
      propertyFactoryService.createProperty(myconfig);
   cproperty.setType("String");

   PropertyValue[] propertyValues = new PropertyValue[1];
   propertyValues[0] =
      propertyFactoryService.createPropertyValue(request, cproperty, serverName);

   DynamicUICtrl dmanagerCtrl =
      dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

   ObjectID newPortletID1 =
      dmanagerCtrl.addSharedPortlet(portletDefinitionID1, null, propertyValues);
   ObjectID newPortletID2 =
      dmanagerCtrl.addSharedPortlet(portletDefinitionID2, null, propertyValues);

      logger.log(Level.FINE, "portlet ID created:"+newPortletID1.getOID());
    } catch ...
```

In some cases, the target portlet might be on a separate page. In this case, the properties are passed using the addPage() method. The target portlet receives the properties only when the page is launched that contains the portlet. If a property value is set multiple times before the page is launched, the value that was set last for the property is passed to the portlets on the page. You can use the URLGeneratorFactoryService interface to redirect the console to the page for the target portlet. This is demonstrated in the GetApps portlet of the PortletContext sample.

```
ObjectID newPageID = dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);
RedirectURLGenerator urlGenerator = urlGeneratorFactoryService.getURLGenerator(request, response);
EngineURL redirectURL = urlGenerator.createPageURL(newPageID);
response.sendRedirect(redirectURL.toString());
```

To use the URLGeneratorFactoryService, the portlet must first perform a JNDI lookup for this portlet service. Refer to the GetApps.java source to see how to obtain a reference to the portlet service.

Receiving properties

To receive properties, the target portlet must provide the `com.ibm.portal.pagecontext.enable` preference parameter in the portlet.xml with a value of true. If the portlet should receive any subsequent updates, the `com.ibm.portal.context.enable` read-only preference should also be set to true. Only String property types are supported and the context is passed as parameters of the action request. The following example shows how the GetApps portlet receives properties passed by the SetServer portlet.

```
public void processAction(ActionRequest request, ActionResponse response)
       throws PortletException, IOException {

        PortletSession ps = request.getPortletSession(true);
        appDeployed = request.getParameter("appDeployed");
        serverName=request.getParameter("servername");

        ps.setAttribute("servername",serverName);
```

```
        ps.setAttribute("appDeployed",appDeployed);
        launchPage(request, response);

    }
```

A target portlet can check `com.ibm.portal.action` parameter of the request during action processing to determine if any properties have been passed. If properties are being passed to the portlet, the value of this parameter is `com.ibm.portal.pagecontext.receive`. For example:

```
String action = req.getParameter(com.ibm.portal.action.name);
if (action!=null && action.equalsIgnoreCase("com.ibm.portal.pagecontext.receive")) {

  // code to get the properties as a parameter on the request

}
```

Launching Eclipse-based help

If your console module contains a portlet that requires a help page, follow the instructions in this topic to implement help for the portlet.

About this task

To implement portlet help, you use the portlet's doHelp() method, the EclipseHelp class, and the help setting in the portlet.xml file. Inside the doHelp() method, link to the target help topic in your Eclipse help plug-in by using the portletHelp() method of the EclipseHelp class. By including a doHelp() method in your portlet and adding the help setting to the portlet.xml file, you enable the portlet help mode. When portlet help mode is enabled, the ? icon is displayed on the portlet title bar. When a user clicks the icon, a separate browser window opens and displays only that help topic.

The Modes sample (modes.war) demonstrates how a console module can provide user assistance or help. This sample is described in "Example: Console module samples" on page 119.

1. Add the doHelp() method to your portlet.
   a. In the portlet Java file, include the following import statement:
      ```
      import com.ibm.portal.help.EclipseHelp;
      ```
   b. Implement the doHelp() method in the portlet.
      ```
      public void doHelp(RenderRequest request, RenderResponse response)
             throws PortletException, IOException {

      }
      ```
   c. Create a String in the deHelp() method to contain the help topic. The syntax for specifying a help topic is *pluginID*/*file_path*/*file_name*, where *pluginID* is the Eclipse help plug-in ID as specified in its plugin.xml file, *file_path* is the sub-directory path (if any), and *file_name* is the filename and extension of the help content file.
      ```
      public void doHelp(RenderRequest request, RenderResponse response)
             throws PortletException, IOException {

         String topic = "com.ibm.isc.help.modes/help_mode.html";
      }
      ```
   d. Call the method EclipseHelp.portletHelp() passing in the request, response, and topic as shown in the code snippet below.
      ```
      public void doHelp(RenderRequest request, RenderResponse response)
             throws PortletException, IOException{

         String topic = "com.ibm.isc.help.modes/help_mode.html";
        EclipseHelp.portletHelp(request, response, topic);

      }
      ```

2. Set help mode in the portlet.xml file.
   a. Use a text editor to open the portlet deployment descriptor (the portlet.xml file). See topics under Developing portlets for more information about creating a portlet.xml file.
   b. Inside the <portlet-app> element, locate the <portlet> element for the portlet that implements the doHelp() method.
3. Inside the <supports> sub-element, add a <portlet-mode> element like the one shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0" id="com.ibm.isclite.samples.Modes"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                                 http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
   <portlet id="Modes_Portlet" >
      <portlet-name>ModesPortlet</portlet-name>
      <display-name>Modes portlet1 (JSR 168)</display-name>
      <display-name xml:lang="en">Modes portlet1 (JSR 168)</display-name>
      <portlet-class>com.ibm.isclite.samples.Modes</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>VIEW</portlet-mode>
         <portlet-mode>EDIT</portlet-mode>
         <portlet-mode>HELP</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>nls.HTML</resource-bundle>
   </portlet>


</portlet-app>
```

4. When packaging the console module, place the Eclipse help plugin in the /help directory of the WAR file. See “Packaging a console module” on page 125 for more information.

Support for bidirectional characters

Integrated Solutions Console supports bidirectional text. If your console module provides support for Hebrew, it must provide two identical resource bundles for the translated Strings. The file name for one resource bundle must be appended with the `_iw` suffix and the other resource bundle must be appended with the `_he` suffix. For example:

```
portlet_iw.properties
portlet_he.properties
```

Developing your first console module

Follow these steps if you are new to console module development.

Before you begin

You should have an integrated development environment (IDE) that supports Java programming setup. See “Setting up the development environment” on page 120 for more information.

The topics in this section take you through the process of creating a simple console module. The sample console module consists of a single portlet which is deployed to a single page as part of the set of sample console modules provided by IBM. Be sure you have successfully deployed the sample console modules before starting.

Developing the portlet

This topic is intended for developers who are unfamiliar with portlet development. A simple Java class and JSP are provided.

Developing the source Java class file

The following example shows the Java code for a portlet in its simplest form.

```
package com.ibm.isclite.samples.basicmodule;

import java.io.*;
import javax.portlet.*;

public class BasicModule extends GenericPortlet {

 public void doView(RenderRequest request, RenderResponse response)
             throws PortletException, IOException {
  // Set the MIME type for the render response
  response.setContentType("text/html");

        // Invoke the JSP to render
  PortletRequestDispatcher rd = getPortletContext().getRequestDispatcher("/jsp/basicView.jsp");
  rd.include(request,response);
 }

}
```

The portlet code must extend the GenericPortlet class and output for the response in the doView() method. Portlets are rendered in different modes. The initial mode when a portlet is called to render is the view mode. The response output is provided by a JSP, which provides markup that can be aggregated into a larger HTML page. The package name in this example is consistent with the console module samples.

Developing the JSP

The following shows a portlet JSP for view mode in its simplest form.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1" session="false" buffer="none"%>
<%@ page import="javax.portlet.*" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portletAPI" %>

<portletAPI:defineObjects />

<%
  PortletPreferences prefs = renderRequest.getPreferences();
  String URL = prefs.getValue("website","");
%>

<p><a name="<portletAPI:namespace/>basicAnchor">Basic contents</a></p>

<p>
  <a href="<%=URL%>" target="_blank">
    <img src='<%=renderResponse.encodeURL(renderRequest.getContextPath() + "/images/logo.gif")%>'
         alt="logo" />
      </a>
</p>
<p><em>Company logo with link.</em></p>
```

The taglib directive specifies the portlet tags from the Java Portlet Specification. The following tags are used in this example.

namespace
Uniquely qualifies named HTML tags or JavaScript functions to prevent clashes with other portlets on the page using the same name.

defineObjects
Makes the RenderRequest, RenderResponse, and PortletConfig objects available to the JSP. This tag is used in this example so that the image can be rendered using the encodeURL() method of the render response, and to allow the JSP to read preferences in the portlet descriptor.

Creating the descriptors for the console module
Before you begin

The following descriptors are required for the console module.

- Web application descriptor (web.xml). This descriptor is required and described by the Java Servlet Specification.
- Portlet application descriptor (portlet.xml). This descriptor is required and described by the Java Portlet Specification. Refer to the console module samples for examples of how to create the portlet.xml.
- Portal topology descriptor (ibm-portal-topology.xml). This descriptor is unique to console modules and defines the portlets and resources in the module, how it is laid out on the page, and how it is accessed from the navigation.
- Portal security descriptor (ibm-portal-security.xml). This descriptor is unique to console modules and defines roles and access to navigation nodes and portlets.

About this task

The topology and security descriptors are the main component of console module packages that distinguish them from other portal application WAR files. The console module samples include these descriptors and demonstrate their features. The sample descriptors provide inline comments that explain the elements and how they are used.

For your first console module, you should use the Visual Designer plugin to the Application Server Toolkit to develop the descriptors. If you prefer to create them manually to become familiar with the structure of the XML, the steps in this topic show how to create simple descriptors that can be used to deploy an application into the samples provided by IBM. After following these steps, you should refer to the samples and the reference topics for the console module schemas for more complete information.

Portal topology descriptor
About this task

To follow these steps, open a file with the name ibm-portal-topology.xml in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into ibm-portal-topology.xml. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-topology
    xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd"
    xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
ibm-portal-topology.xsd">

   <application-definition appID="com.ibm.isclite.samples.basicModule" version="6.1">
```

```
   </application-definition>
   <PAA-ref>com.ibm.isclite.samples.PageLayout</PAA-ref>

</ibm-portal-topology>
```

- The elements in this step provide the top level content of the topology descriptor.
- Most of the content for this descriptor will be placed inside the <application-definition/> element.
- The value of the **appID** attribute is consistent with the naming convention used by the sample console modules. The **version** attribute matches the version used by the sample console modules.
- The <PAA-ref/> element specifies the **appID** of the page layout sample, which is the lead application to which this application will be deployed.
- The following convention is used for unique name for all of the elements in the descriptor.

```
namespace + element_type + identifier
```

2. Create a title for your application. The title is the first element in the application definition element.

```
<title>
      <base:nls-string lang="en">My basic module</base:nls-string>
</title>
```

3. Place the component tree after the <requires/> element as shown.

```
<component-tree uniqueName="com.ibm.isclite.samples.basicModule.appTree">
   <portlet-definition uniqueName="com.ibm.isclite.samples.basicModule.portletDefinition.A">
      <title>
         <base:nls-string lang="en">My basic module</base:nls-string>
      </title>
      <resource-link name="basicModule"
                     portletApplication="com.ibm.isclite.samples.basicmodule.BasicModule.01a"
                     type="static"/>
   </portlet-definition>
   <portlet-entity   uniqueName="com.ibm.isclite.samples.basicModule.portletEntity.A"
                     portletDefinitionRef="com.ibm.isclite.samples.basicModule.portletDefinition.A">
      <title>
         <base:nls-string lang="en">My basic module</base:nls-string>
      </title>
   </portlet-entity>
</component-tree>
```

- The **name** attribute of the <resource-link/> element must specify the <portlet-name/> from the portlet deployment descriptor.
- The **portletApplication** attribute of the <resource-link/> element must specify the **id** attribute of the <portlet-app/> element from the portlet deployment descriptor.
- The **portletDefinitionRef** attribute of the <portlet-entity/> element references the unique name of the <portlet-definition/>. Subsequent elements in this descriptor use the unique name of the <portlet-entity/> to invoke the portlet. Multiple portlet entities can point to the same portlet definition. For the purpose of this exercise, a single portlet definition is provided with a single portlet entity.
- The portlet title that is rendered in this example is derived from the title in the portlet entity. If that value is not provided, the portlet title is rendered by the title provided by the <window/> element in the layout tree. If that value is also not provided, the title is obtained from the <portlet-name/> element in the portlet.xml for the portlet indicated by the <resource-link/>.

4. Place the layout tree after the component tree as shown.

```
<layout-tree>
   <layout-element uniqueName="com.ibm.isclite.samples.basicModule.layoutElement.A">
      <title>
         <base:nls-string lang="en">My basic module</base:nls-string>
      </title>
      <simple-container orientation="row"
                        uniqueName="com.ibm.isclite.samples.basicModule.container.A">
         <window uniqueName="com.ibm.isclite.samples.basicModule.window.A">
            <title>
               <base:nls-string lang="en">My basic module</base:nls-string>
```

```
            </title>
            <component-definition-ref>
               com.ibm.isclite.samples.basicModule.portletEntity.A
            </component-definition-ref>
         </window>
      </simple-container>
   </layout-element>
</layout-tree>
```

- Each layout element in the layout tree defines the layout of a page in the console.
- Each page can have nested containers, and each simple container can have multiple windows. Each simple container has either a row or column orientation. The nesting of these containers can create a complex table structure for the page layout.
- The <component-definition-ref/> element specifies the unique name of the portlet entity that provides the content of this window.

5. Create the navigation element after the layout tree as shown.

```
<navigation-element uniqueName="com.ibm.isclite.samples.navigationElement.basicModule"
            layoutElementRef="com.ibm.isclite.samples.basicModule.layoutElement.A">
   <title>
      <base:nls-string lang="en">Basic Module Sample</base:nls-string>
   </title>
   <preference name="ProductFilter">
      <base:value>
         <base:string>com.ibm.isclite.samples.PageLayout</base:string>
      </base:value>
   </preference>
   <parent-tree parentTreeRef="com.ibm.isclite.samples.navigationElement.Samples"/>
</navigation-element>
```

- Each navigation element defines a node in the console navigation. If you nest the navigation elements, it creates a hierarchical tree structure that is reflected in the console navigation. For the purpose of this exercise, only a single navigation element is provided. The **layoutElementRef** attribute indicates the unique name of the layout element that is rendered on the page that is displayed when this element is selected by the console user.
- The content of the `ProductFilter` preference indicates the **appID** of the lead application. This is used to render this navigation element when the lead product is selected by the console user for filtering the navigation.
- The content of the **parentTreeRef** element indicates the unique name of the parent navigation element for this element.

The following figure shows how this navigation element is rendered with the page layout sample. In this figure, the user has selected `Integrated Solutions Console Sample` from the **View** selection list to filter the navigation.

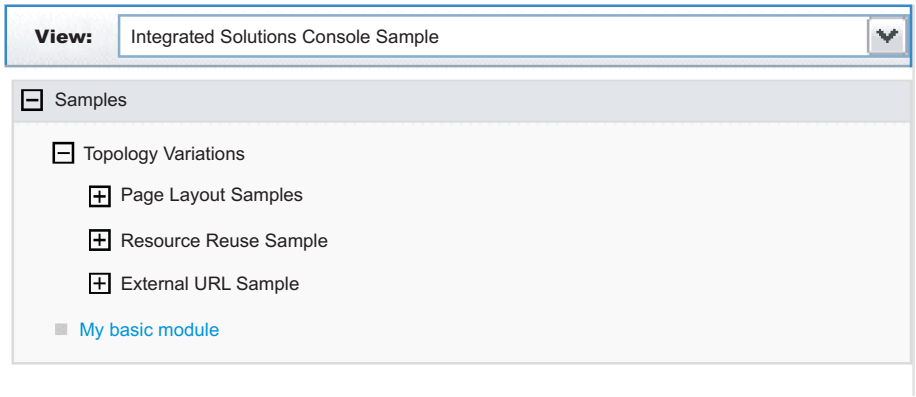


Figure 4. Sample console module navigation

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Portal security descriptor

Before you begin

To follow these steps, open a file with the name ibm-portal-security.xml in a text or XML editor and save each change at the end of each step.

1. Copy the following elements into ibm-portal-security.xml. Some information is split on multiple lines for printing purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<ibm-portal-security
     xmlns="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd"
     xmlns:base="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-topology.xsd
ibm-portal-topology.xsd
         http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-base.xsd ibm-portal-base.xsd
         http://www.ibm.com/websphere/appserver/schemas/6.0/ibm-portal-security.xsd ibm-portal-security.xsd">

  </ibm-portal-security>
```

2. Add the <application-role/> element as the content of the <ibm-portal-security/> element.

```
<application-role uniqueName="monitor">


</application-role>
```

   This sample identifies the administrative role, `monitor`, for access rights. See Portal security schema for a complete description of this element and the uniqueName attribute.

3. Add the following <portal-role/> elements as the content of the application role.

```
<portal-role object-ref="com.ibm.isclite.samples.navigationElement.basicModule"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.navigationElement.A"/>
<portal-role object-ref="com.ibm.isclite.samples.basicModule.portletEntity.A"
             role-type="Privileged User"/>
```

   This sample assigns permission to a user with monitor access to view two navigation elements and to view, edit, and access help for one portlet. See Portal security schema for a complete description of this element and its attributes.

What to do next

When you have finished, save and close the file and store in the /WEB-INF directory of your console module project.

Packaging a console module

Before you begin

Before you start this task, be sure your Java sources compiles without errors and the XML descriptors have been validated against the corresponding schemas.

About this task

An Integrated Solutions Console module must be packaged as a WAR file for deployment. For best results, use a comprehensive development tool such as Rational Application Developer. Otherwise, you can follow these steps to package your console module resources into the WAR file.

1. Create a working directory to collect all of the files that need to be packaged in the WAR file.
2. Organize your portlets and other resource files in the prescribed directory structure for portlet WAR files using the following steps:
   a. Place the following deployment descriptors in the directory /WEB-INF directory:
      - ibm-portal-topology.xml (topology descriptor)
      - ibm-portal-security.xml (security descriptor)
      - portlet.xml (portlet descriptor)
      - web.xml (Web application descriptor)

   b. Place individual Java classes for your portlets in the directory /WEB-INF/classes. Use a directory structure that reflects the fully-qualified class name of your portlet, for example, /com/ibm/isclite/samples/basicModule.
   c. Put JAR files in the directory /WEB-INF/lib.
   d. If the console module supports multiple languages, place the properties file that contains the translated strings (the resource bundles) in the directory /WEB-INF/classes/nls.
   e. Put any JavaServer Pages that must be served directly by the client in a directory that is not under the /WEB-INF directory (such /jsp). Resources in the /WEB-INF directory are protected and are not URL-addressable. JavaServer Pages that are accessed only by the portlets in the console module can be placed in the /WEB-INF path.
   f. If the console module has help files, copy the Eclipse help plug-in files to the directory /help/*plugin_name*, where *plugin_name* is the name of the Eclipse help plug-in, such as com.ibm.myprod.help.doc.
3. Use the Java JAR utility to create the WAR file. For instructions, see the Java Tutorial at http://java.sun.com/docs/books/tutorial/jar/basics/index.html . You can also use an assembly tool like Rational Application Developer to create the WAR file for the console module.

Deploying a console module

After you create the WAR file for a console module, you can deploy the console module to a system where the runtime for Integrated Solutions Console is installed and then test that the console module works correctly. Only one instance of a console module can be deployed on an Integrated Solutions Console installation. This rule applies even if the versions of the console module are different.

About this task

If you need to update the version of a console module, you must perform these steps.

1. Remove the console module.

2. Update the module with the required changes.
3. Reissue the deploy command.

To deploy a console module on a system where the runtime for Integrated Solutions Console is installed, perform the following procedure:

1. Go to the machine where the Integrated Solutions Console runtime is installed. You can also use a machine that has remote access to the drive where the runtime is installed.
2. Copy the console module WAR file to the *app_server_root*/systemApps/isclite.ear directory. As an alternative, you can extract the WAR file into a subdirectory of this location using the WAR file name for the subdirectory. For example, you would extract pagelayout.war to the *app_server_root*/systemApps/isclite.ear/pagelayout.war directory.
3. Grant the QEJBSVR profile the authority needed to complete the task. From an i5/OS command prompt, issue this command:

   ```
   CHGAUT OBJ('path_to_war') USER(QEJBSVR) DTAAUT(*RWX) OBJAUT(*ALL)
   ```
4. Start the wsadmin scripting client. If the server is not running, use the -conntype NONE option to run in local mode.
5. At the **wsadmin** command prompt, enter the command syntax. The following example presents the syntax:

   ```
   $AdminApp update isclite modulefile {
   -operation add
   -contents path_to_war
   -contenturi war_filename
   -custom paavalidation=true
   -custom containWebServices=true|false
   -usedefaultbindings -contextroot /context
   -MapWebModToVH {{.* .* admin_host}}
   -MapModulesToServers {{.*  .*.war,.*  WebSphere:cell=cellname,node=nodename,server=servername}}}
   ```

   Note:

   - The values of the -contents and -contenturi options are case sensitive and should match the war file name that you are ready to deploy.
   - Specifying `-custom paavalidation=true` causes the deployment to validate the XML descriptors for the console module package before deployment. This is optional.
   - Set the option `-custom containWebServices=true` only if the package contains a Web service. This setting invokes more deployment processing to locate and setup the Web service. The default for this option is true. If the package does not contain any Web services, be sure to set this option to false for more efficient deployment.
   - Use the -MapModulesToServers option when deploying to an agent profile.

   For example:

   ```
   $AdminApp update isclite modulefile {
   -operation add
   -contents app_server_root/systemApps/isclite.ear/pagelayout.war
   -contenturi pagelayout.war
   -custom paavalidation=true
   -custom containWebServices=false
   -usedefaultbindings  -contextroot /ibm/pagelayout
   -contextroot /pagelayout
   -MapWebModToVH {{.* .* admin_host}}
   -MapModulesToServers {{.*  .*.war,.*  WebSphere:cell=cell_A,node=node_B,server=server_C}}}
   ```

   The following messages are displayed if the deployment is successful.

   ```
   Update of isclite has started.
   ADMA5009I: An application archive is extracted at profile_root/wstemp/wstemp/app_10793d84b8e/ext/pagelayout.war
   ADMA5064I: FileMergeTask completed successfully for isclite.
   ADMA5005I: The application isclite is configured in the WebSphere Application Se
   rver repository.
   ```

```
ADMA5005I: The application isclite is configured in the WebSphere Application Se
rver repository.
ADMA5110I: The application isclite is installed as a hidden application and will
 not be exposed via administrative interfaces such as GUI client, wsadmin or MBe
an Java API.  In order to perform management operations on this application, the
 application name must be known.
CWLAA10001I: The Integrated Solutions Console module was deployed successfully.
ADMA5005I: The application isclite is configured in the WebSphere Application Se
rver repository.
ADMA5011I: The cleanup of the temp directory for application isclite is complete
.
Update of isclite has ended.
```

6. After the deployment has completed successfully, save your work by issuing the following command:

```
$AdminConfig save
```

Results

If the deployment is successful, the WAR file contents are extracted to the /systemApps/isclite.war directory and the module metadata is copied to the following directory:

profile_root/config/cells/*cellname*/applications/isclite.ear/deployments/isclite

To view your console module, you must log back in to the console. Make sure the navigation is updated correctly. If so, select the new navigation nodes verify that the new pages and console modules are rendered correctly.

Testing a console module

About this task

Use the following checklist to test your console module after it has been deployed.

1. Log in to Integrated Solutions Console as an administrator.
2. If the module includes an <about-page/> element in the topology descriptor, verify that your suite and its version are listed in the table on the Welcome page. Also confirm that the product is listed in the **View** selection list for the navigation.
3. Verify that the console navigation tree contains all of the organizational nodes and pages that were defined. Verify that each node is at the correct level in the tree.
4. For each page, verify that the portlets on the page work correctly. If a portlet on the page launches a new page in work area, verify that the launching works correctly.
5. To test the access permissions specified in the console module descriptor, logout and then log in as a different user to verify that the correct pages are accessible.
6. If the console module included a help plug-in, make sure you can launch the help.
7. If the console module contains errors, refer to the SystemOut.log and SystemErr.log for any errors or messages. These logs are located in the following directory:

   profile_root/logs/server1

Removing a console module

About this task

Follow these steps to delete a console module.

1. Ensure that the server is started. You do not need to log in to the console.
2. Start the wsadmin scripting client.
3. Enter the following command from the wsadmin command prompt.

```
$AdminApp update isclite modulefile {-operation delete -contenturi module_file_name.war}
```

4. Save your configuration changes.

```
$AdminConfig save
```

5. Exit the wsadmin command environment.

```
quit
```

Adding advanced API features

The class files for console modules are developed using the Java Portlet Specification. Integrated Solutions Console includes additional APIs for launching pages, passing properties to other portlets, and launching Eclipse-based help. This section provides information about the APIs available to console modules.

Developing portlets

Console modules are portlets that run in the portlet container of WebSphere Application Server.

Integrated Solutions Console supports console modules written to the Java Portlet Specification. A packaged console module WAR file contains one or more portlets or reuses portlets from other modules. According to the JSR 168 specification, a *portlet* is a Java technology based Web component, managed by a portlet container, that processes requests and generates content that contributes to the overall portal page. Portlets share many of the same characteristics as servlets. However, portlets also exhibit differences. For example, portlets cannot set HTTP headers and can contribute only markup fragments to the portal page. The portlet specification leverages the functionality of the servlet specification as much as possible, and many parts of the Java Portlet Specification have been modeled after the servlet specification.

The application server includes a JSR 168-compliant portlet container to support deploying and running console modules in Integrated Solutions Console. The JSR 168 portlet container relies on the Java Platform, Enterprise Edition (Java EE) architecture implemented by the application server. Consequently, an Integrated Solutions Console module and its portlets are packaged and deployed similar to Java EE Web applications. In addition to the Web application descriptor and the portlet descriptor, the WAR file for an Integrated Solutions Console module includes additional descriptors needed to deploy the application to a console installation.

The following Web resources can useful for learning more about portlet development.

Resource	Web site URL
Portlet Specification 1.0	http://jcp.org/jsr/detail/168.jsp
Portlet Specification 2.0	http://jcp.org/en/jsr/detail?id=286
WebSphere Portal Zone	http://www.ibm.com/developerworks/websphere/zones/portal/
Best practices: Developing portlets using JSR 168 and WebSphere Portal	http://www.ibm.com/developerworks/websphere/library/techarticles/0403_hepper/0403_hepper.html
WebSphere Portal external newsgroup	news://news.software.ibm.com/ibm.software.websphere.portal-server
Develop a Faces JSR 168 portlet using IBM Rational Application Developer 6.0 for autonomic computing	http://www.ibm.com/developerworks/edu/ac-dw-ac-jsfisc-i.html
Introduction to JSR 168 - The Portlet Specification	http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/index.html

Launching pages

Typically, users of the console launch pages using the links in the console navigation. However, portlets in a console module also launch pages directly using the Dynamic UI Manager API.

The DynamicUI Manager APIs are provided by 3 services: PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService. To use these services the portlet must first perform a JNDI lookup to get the PortletServiceHome object which you then can use to get the actual service interface object. This is demonstrated in the GetApps portlet of the PortletContext sample.. The following code is placed in a `try` block of the portlet's init() method.

Some information is split on multiple lines for printing purposes.

```
Context ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
// get propertyfactory service

propertyFactoryServiceHome = (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.
propertybroker.service.PropertyBrokerService");
if (propertyFactoryServiceHome == null)
   logger.log(Level.FINE, "no PropertyBrokerService propertyFactoryServiceHome");
else
   propertyFactoryService =

      (PropertyFactory)propertyFactoryServiceHome.getPortletService(com.ibm.portal.
propertybroker.service.PropertyBrokerService.class);
logger.log(Level.FINE, "propertyBrokerService="+propertyFactoryService);
// get dynamicUIManagerFactory service
dynamicUIManagerFactoryServiceHome =
   (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.DynamicUIManagerFactoryService");
if (dynamicUIManagerFactoryServiceHome == null)
   logger.log(Level.FINE, "no dynamicUIManagerFactoryService propertyFactoryServiceHome");
else
   dynamicUIManagerFactoryService =

      (DynamicUIManagerFactoryService)dynamicUIManagerFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.DynamicUIManagerFactoryService.class);
logger.log(Level.FINE, "dynamicUIManagerFactoryService="+dynamicUIManagerFactoryService);
// get urlGeneratorFactory service
urlGeneratorFactoryServiceHome =
   (PortletServiceHome)ctx.lookup("iscportletservice/com.ibm.portal.portlet.service.URLGeneratorFactoryService");
if (urlGeneratorFactoryServiceHome == null)
   logger.log(Level.FINE, "no urlGeneratorFactoryService propertyFactoryServiceHome");
else
   urlGeneratorFactoryService =

      (URLGeneratorFactoryService) urlGeneratorFactoryServiceHome.getPortletService
(com.ibm.portal.portlet.service.URLGeneratorFactoryService.class);
logger.log(Level.FINE, "urlGeneratorFactoryService="+urlGeneratorFactoryService);
```

A portlet can launch a new page using the dynamicUICtrl interface, passing the object ID of the page definition for the new page. The object ID is determined by performing a JNDI lookup on the unique name of the navigation element that launches the page. If you are planning to launch a navigation element, the navigation node must point to a layout element. In other words the navigation element must contain the l**ayoutElementRef** attribute. The addPage() method returns the object ID of the page instance, which should be saved by the portlet to create a URL and launch the page. The following excerpt from the PortletContext sample demonstrates how this works. The portlet GetApps.java includes a launchPage() method with this code.

```
Context ctx;
ObjectID pageDefinitionID = null;
String pagename="com.ibm.isclite.portletcontext.navigationElement.B";
logger.log(Level.FINE, "launch page=" + pagename);

PortletSession ps  = request.getPortletSession(false);
try {
    ctx = new InitialContext();
    pageDefinitionID =(ObjectID)ctx.lookup("portal:uniquename/"+pagename);
} catch (NamingException ne) {
    return;
}
```

```
try {
    PropertyController cproperty = propertyFactoryService.createProperty(myconfig);
    PropertyController cproperty2 = propertyFactoryService.createProperty(myconfig);

    cproperty.setType("String");
    cproperty.setName("servername");
    cproperty2.setType("String");
    cproperty2.setName("appDeployed");
    PropertyValue[] propertyValues = new PropertyValue[2];
    propertyValues[0] =
        propertyFactoryService.createPropertyValue(request, cproperty, serverName);
    propertyValues[1] =
        propertyFactoryService.createPropertyValue(request, cproperty2, appDeployed);

    DynamicUICtrl dmanagerCtrl =
       dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

    ObjectID newPageID =
        dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);

    RedirectURLGenerator urlGenerator =
        urlGeneratorFactoryService.getURLGenerator(request, response);

    EngineURL redirectURL = urlGenerator.createPageURL(newPageID);

    response.sendRedirect(redirectURL.toString());
} catch  ...
```

- The `pagename` String is created with the unique name of the navigation element (from the topology descriptor) that launches the page.
- The object ID is found by performing a JNDI lookup on this Context instance using this String:

  ```
  "portal:uniquename/"+pagename
  ```

  The String `portal:uniquename/` is required to obtain the object ID for the navigation element.
- Properties are created to be passed to the portlets on the page. The property names are `servername` and `appDeployed`.
- In addition to the request and response, you must include the String `isc.tasklaunch` as the configuration name on the getDynamicUICtrl() method.
- Use the addPage() method of the new DynamicUICtrl instance to create and instance of the page. Any properties that need to be passed to portlets on the new page must be provided in this method. The addPage() method does not launch the page, but it returns the object ID of the page instance. The next two methods, getURLGenerator() and createPageURL(), are used to construct the URL that opens the page. The URL contains property names and values if they have been provided.
- The URL to open the page is provided on the sendRedirect() method of the response.

Passing properties to other portlets

This topic describes how portlets can exchange properties. You should be familiar with the concepts for launching console pages before reading this topic.

Information can be provided to portlets on the same page using the addSharedPortlet() method. Property values are of the type com.ibm.portal.propertybroker.property.PropertyValue. The example in “Launching pages” on page 129 shows these properties passed to the page using an instance of the PropertyValue interface (`propertyValues`) and shows how that object was created using the PropertyFactory interface.

The SetServer.java source from the PortletContext sample shows how to pass properties to a portlet on the same page. It is assumed that the portlet has already performed a JNDI lookup to determine that the PropertyFactory, DynamicUIManagerFactoryService, and URLGeneratorFactoryService services are available. This is described in “Launching pages” on page 129

- In this example, the portlet that is the source of the properties needs the object ID of the target portlet. The object ID is determined by performing a JNDI lookup using the values from the portlet's <resource-link/> element in the topology descriptor.
- The property, cproperty, is created using the PropertyController interface and the createProperty() method of the PropertyFactory interface. The data type for the property must be a String.
- Prior to sending the property, the source portlet creates an instance of the DynamicUICtrl interface, passing the string ″isc.tasklaunch″ as the configuration name.
- The addSharedPortlet() method sends the properties to the target portlet on the request. After the action phase, the target portlet updates the output for the response.

```
Context ctx = null;
ObjectID portletDefinitionID1 = null;
ObjectID portletDefinitionID2 = null;

String portletname1="com.ibm.isclite.samples.PortletContext/PortletGetPerformance";
String portletname2="com.ibm.isclite.samples.PortletContext/PortletGetApps";

PortletSession ps  = request.getPortletSession(false);

try
   {
      ctx = new InitialContext(com.ibm.portal.jndi.Constants.ENV);
      portletDefinitionID1 =
         (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname1);
      portletDefinitionID2 =
         (ObjectID)ctx.lookup("portal:config/portletdefinition/"+portletname2);
   } catch (NamingException ne)
     {
        logger.log(Level.FINE, "portletdefinitionID not found - Naming exception:"+ne.getMessage());
        return;
     }
logger.log(Level.FINE, "portletdefinitionID="+portletDefinitionID1.toString());

try
{
   PropertyController cproperty =
      propertyFactoryService.createProperty(myconfig);
   cproperty.setType("String");

   PropertyValue[] propertyValues = new PropertyValue[1];
   propertyValues[0] =
      propertyFactoryService.createPropertyValue(request, cproperty, serverName);

   DynamicUICtrl dmanagerCtrl =
      dynamicUIManagerFactoryService.getDynamicUICtrl(request, response, "isc.tasklaunch");

   ObjectID newPortletID1 =
      dmanagerCtrl.addSharedPortlet(portletDefinitionID1, null, propertyValues);
   ObjectID newPortletID2 =
      dmanagerCtrl.addSharedPortlet(portletDefinitionID2, null, propertyValues);

      logger.log(Level.FINE, "portlet ID created:"+newPortletID1.getOID());
    } catch ...
```

In some cases, the target portlet might be on a separate page. In this case, the properties are passed using the addPage() method. The target portlet receives the properties only when the page is launched that contains the portlet. If a property value is set multiple times before the page is launched, the value that was set last for the property is passed to the portlets on the page. You can use the URLGeneratorFactoryService interface to redirect the console to the page for the target portlet. This is demonstrated in the GetApps portlet of the PortletContext sample.

```
ObjectID newPageID = dmanagerCtrl.addPage(pageDefinitionID, null, propertyValues);
RedirectURLGenerator urlGenerator = urlGeneratorFactoryService.getURLGenerator(request, response);
EngineURL redirectURL = urlGenerator.createPageURL(newPageID);
response.sendRedirect(redirectURL.toString());
```

To use the URLGeneratorFactoryService, the portlet must first perform a JNDI lookup for this portlet service. Refer to the GetApps.java source to see how to obtain a reference to the portlet service.

Receiving properties

To receive properties, the target portlet must provide the `com.ibm.portal.pagecontext.enable` preference parameter in the portlet.xml with a value of true. If the portlet should receive any subsequent updates, the `com.ibm.portal.context.enable` read-only preference should also be set to true. Only String property types are supported and the context is passed as parameters of the action request. The following example shows how the GetApps portlet receives properties passed by the SetServer portlet.

```
public void processAction(ActionRequest request, ActionResponse response)
       throws PortletException, IOException {

        PortletSession ps = request.getPortletSession(true);
        appDeployed = request.getParameter("appDeployed");
        serverName=request.getParameter("servername");

        ps.setAttribute("servername",serverName);
        ps.setAttribute("appDeployed",appDeployed);
        launchPage(request, response);

    }
```

A target portlet can check `com.ibm.portal.action` parameter of the request during action processing to determine if any properties have been passed. If properties are being passed to the portlet, the value of this parameter is `com.ibm.portal.pagecontext.receive`. For example:

```
String action = req.getParameter(com.ibm.portal.action.name);
if (action!=null && action.equalsIgnoreCase("com.ibm.portal.pagecontext.receive")) {

  // code to get the properties as a parameter on the request

}
```

Launching Eclipse-based help

If your console module contains a portlet that requires a help page, follow the instructions in this topic to implement help for the portlet.

About this task

To implement portlet help, you use the portlet's doHelp() method, the EclipseHelp class, and the help setting in the portlet.xml file. Inside the doHelp() method, link to the target help topic in your Eclipse help plug-in by using the portletHelp() method of the EclipseHelp class. By including a doHelp() method in your portlet and adding the help setting to the portlet.xml file, you enable the portlet help mode. When portlet help mode is enabled, the ? icon is displayed on the portlet title bar. When a user clicks the icon, a separate browser window opens and displays only that help topic.

The Modes sample (modes.war) demonstrates how a console module can provide user assistance or help. This sample is described in "Example: Console module samples" on page 119.

1. Add the doHelp() method to your portlet.
   a. In the portlet Java file, include the following import statement:
      ```
      import com.ibm.portal.help.EclipseHelp;
      ```
   b. Implement the doHelp() method in the portlet.

```
public void doHelp(RenderRequest request, RenderResponse response)
       throws PortletException, IOException {

}
```

c. Create a String in the deHelp() method to contain the help topic. The syntax for specifying a help topic is *pluginID*/*file_path*/*file_name*, where *pluginID* is the Eclipse help plug-in ID as specified in its plugin.xml file, *file_path* is the sub-directory path (if any), and *file_name* is the filename and extension of the help content file.

```
public void doHelp(RenderRequest request, RenderResponse response)
       throws PortletException, IOException {

   String topic = "com.ibm.isc.help.modes/help_mode.html";
}
```

d. Call the method EclipseHelp.portletHelp() passing in the request, response, and topic as shown in the code snippet below.

```
public void doHelp(RenderRequest request, RenderResponse response)
       throws PortletException, IOException{

   String topic = "com.ibm.isc.help.modes/help_mode.html";
  EclipseHelp.portletHelp(request, response, topic);

}
```

2. Set help mode in the portlet.xml file.
   a. Use a text editor to open the portlet deployment descriptor (the portlet.xml file). See topics under Developing portlets for more information about creating a portlet.xml file.
   b. Inside the <portlet-app> element, locate the <portlet> element for the portlet that implements the doHelp() method.
3. Inside the <supports> sub-element, add a <portlet-mode> element like the one shown in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
             version="1.0" id="com.ibm.isclite.samples.Modes"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                                 http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
   <portlet id="Modes_Portlet" >
      <portlet-name>ModesPortlet</portlet-name>
      <display-name>Modes portlet1 (JSR 168)</display-name>
      <display-name xml:lang="en">Modes portlet1 (JSR 168)</display-name>
      <portlet-class>com.ibm.isclite.samples.Modes</portlet-class>
      <expiration-cache>0</expiration-cache>
      <supports>
         <mime-type>text/html</mime-type>
         <portlet-mode>VIEW</portlet-mode>
         <portlet-mode>EDIT</portlet-mode>
         <portlet-mode>HELP</portlet-mode>
      </supports>
      <supported-locale>en</supported-locale>
      <resource-bundle>nls.HTML</resource-bundle>
   </portlet>


</portlet-app>
```

4. When packaging the console module, place the Eclipse help plugin in the /help directory of the WAR file. See “Packaging a console module” on page 125 for more information.

Support for bidirectional characters

Integrated Solutions Console supports bidirectional text. If your console module provides support for Hebrew, it must provide two identical resource bundles for the translated Strings. The file name for one resource bundle must be appended with the `_iw` suffix and the other resource bundle must be appended with the `_he` suffix. For example:

```
portlet_iw.properties
portlet_he.properties
```

Chapter 3. EJB applications

Task overview: Using enterprise beans in applications

This article provides an overview of the tasks you must perform to use enterprise beans in a Java-based application.

About this task

Use the following steps to develop an Enterprise JavaBeans (EJB) application:

1. **EJB 3.0 beans:** Design a Java Platform, Enterprise Edition (Java EE) application and the enterprise beans that it needs.
2. **EJB 2.x beans:** Design a Java 2 Platform, Enterprise Edition (J2EE) application and the enterprise beans that it needs.
3. Develop any enterprise beans that your application uses.
4. Prepare for assembly. For your EJB 2.x-compliant entity beans, decide on an appropriate access intent policy.
5. Assemble the beans into one or more EJB modules using one of the assembly tools. This process includes setting security. For your EJB 2.x-compliant entity beans, you might also want to designate container-managed persistence (CMP) sequence groups.
6. **EJB 3.0 beans:** Assemble the beans into one or more EJB 3.0 modules using one of the assembly tools.
7. Assemble the modules into a J2EE application using the assembly tool.
8. For a given application server, update the EJB container configuration if needed for the application to be deployed.
9. For a given application server, update the EJB container configuration if needed for the application to be deployed, and determine if you want to batch commands or defer commands for container-managed persistence.
10. Deploy the application in an application server.
11. Test the modules.
    - As needed, debug problems with the container.
    - Debug access problems.
12. Assemble the production application using one of the assembly tools
13. Deploy the application to a production environment.
14. Manage the application:
    a. Manage installed EJB modules. After an application has been installed, you can manage its EJB modules individually through assembly tools.
    b. Manage other aspects of the Java application.
15. Update the module and redeploy it using one of the assembly tools.
16. Tune the performance of the application. See Best practices for developing enterprise beans.

Enterprise beans

An enterprise bean is a Java component that can be combined with other resources to create Java applications. There are three types of enterprise beans, *entity* beans, *session* beans, and *message-driven* beans.

All beans reside in Enterprise JavaBeans (EJB) containers, which provide an interface between the beans and the application server on which they reside.

EJB 2.1 and earlier versions of the specification define entity beans as a means to store permanent data, so they require connections to a form of persistent storage. This storage might be a database, an existing legacy application, a file, or another type of persistent storage.

The EJB 3.0 specification deprecates EJB 1.1-style entity beans. The Java Persistence API (JPA) specification is intended to replace the deprecated enterprise beans. While the JPA replacement is called an entity class, it should not be confused with entity enterprise beans. A JPA entity is not an enterprise bean and is not required to run in an EJB container.

Session beans typically contain the high-level and mid-level business logic for an application. Each method on a session bean performs a particular high-level operation. For example, submitting an order or transferring money between accounts. Session beans often invoke methods on entity beans in the course of their business logic.

Session beans can be either *stateful* or *stateless*. A stateful bean instance is intended for use by a single client during its lifetime, where the client performs a series of method calls that are related to each other in time for that client. One example is a *shopping cart* where the client adds items to the cart over the course of an online shopping session. In contrast, a stateless bean instance is typically used by many clients during its lifetime, so stateless beans are appropriate for business logic operations that can be completed in the span of a single method invocation. Stateful beans should be used only where absolutely necessary. Using stateless beans improves the ability to debug, maintain, and scale the application.

The EJB 3.0 specification supports stateless and stateful session beans. They follow a simple pattern such as:

- Define the business interface.
- Define the class that implements it.
- Add metadata with annotations or with XML deployment descriptors.

The end result of a simple EJB 3.0 stateful session bean looks like the following:

```
package ejb3demo;

@Stateful
public class Cart3Bean implements ShoppingCart {
    private ArrayList contents = new ArrayList();

public void addToCart (Object o) {
  contents.add(o);
    }

public Collection getContents() {
    return contents;

    }
}
```

EJB components can use annotations such as @EJB and other injectable @Resource references if the module is an EJB 3.0 module.

Web application clients and application clients can use deployment descriptor-defined EJB references. If the reference is for an EJB 3.0 session bean without a home interface, the reference should be defined with a null <home> or <local-home> setting in the deployment descriptor.

Web application clients and application clients can also use @EJB injections for references to EJB session beans within the same enterprise archive (EAR) file, but the binding must either use the AutoLink support within the container or the annotation must use the name of the reference that is defined by the deployment descriptor and bound when the application is installed. For more information about AutoLink, see the topic, ″EJB 3.0 application bindings support.″

Message-driven beans enable asynchronous message servicing.

- The EJB container and a Java Message Service (JMS) provider work together to process messages. When a message arrives from another application component through JMS, the EJB container forwards it through an onMessage method call to a message-driven bean instance, which then processes the message. In other respects, message-driven beans are similar to stateless session beans.
- The EJB container and a Java Connector Architecture (JCA) resource adapter work together to process messages from an enterprise information system (EIS). When a message arrives from an EIS, the resource adapter receives the message and forwards it to a message-driven bean, which then processes the message. The message-driven bean is provided services such as transaction support by the EJB container in the same way that other enterprise beans are provided service.

Beans that require data access use *data sources*, which are administrative resources that define pools of connections to persistent storage mechanisms.

EJB modules

An Enterprise JavaBeans (EJB) module is used to assemble one or more enterprise beans into a single deployable unit. An EJB module is stored in a standard Java archive (JAR) file.

An EJB module contains the following:

- One or more deployable enterprise beans.
- A deployment descriptor, stored in an Extensible Markup Language (XML) file. This file declares the contents of the module, defines the structure and external dependencies of the beans in the module, and describes how the beans are to be used at run time.

  It is not necessary to use XML deployment descriptors in EJB 3.0 modules, although XML descriptors are supported. Instead of deployment descriptors, you can use annotations to provide component metadata.

You can deploy an EJB module as a stand alone application, or combine it with other EJB modules or with Web modules to create a Java application. An EJB module is installed and run in an enterprise bean container.

If you want to package an EJB 3.0 module with a deployment descriptor, there are several ways to do it. You can package an EJB 3.0 module with an EJB 3.0 style session and/or message-driven beans exclusively; with an EJB 2.1 style session and/or message-driven beans exclusively, or a combination of 2.1 and 3.0 style beans. The XML deployment descriptor must be a Version 3.0 deployment descriptor. It is required that 2.1 entity beans are packaged in modules with 2.1 deployment descriptors.

EJB modules that contain EJB 3.0 beans must be at the EJB 3.0 specification level when running on the product. To set the EJB module to support EJB 3.0 beans, you can set the ejb-jar.xml deployment descriptor level to 3.0, or you can make sure that the module does not contain an ejb-jar.xml deployment descriptor. If the module level is EJB 2.1 or earlier, no EJB 3.0 functions, including annotation scanning or resource injection is performed at runtime.

For more information about packaging and deployment of EJB 3.0 beans, see the topic EJB 3.0 module packaging overview.

EJB containers

An Enterprise JavaBeans (EJB) container provides a run-time environment for enterprise beans within the application server. The container handles all aspects of an enterprise bean's operation within the application server and acts as an intermediary between the user-written business logic within the bean and the rest of the application server environment.

One or more EJB modules, each containing one or more enterprise beans, can be installed in a single container.

The EJB container provides many services to the enterprise bean, including the following:
- Beginning, committing, and rolling back transactions as necessary.
- Maintaining pools of enterprise bean instances ready for incoming requests and moving these instances between the inactive pools and an active state, ensuring that threading conditions within the bean are satisfied.
- Most importantly, automatically synchronizing data in an entity bean's instance variables with corresponding data items stored in persistent storage.

By dynamically maintaining a set of active bean instances and synchronizing bean state with persistent storage when beans are moved into and out of active state, the container makes it possible for an application to manage many more bean instances than could otherwise simultaneously be held in the application server's memory. In this respect, an EJB container provides services similar to virtual memory within an operating system.

WebSphere Application Server provides significant flexibility in the management of database data with entity beans. The Entity EJBs Activate at and Load at configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean caching Options A, B or C, as specified in the EJB 1.1 specifications. You can configure these settings with assembly tools. To read more about how to use the assembly tools see the assembly tool information center.

Option A provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, Option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. Option B provides more aggressive caching of Entity EJB object instances, which can result in improved performance over Option C, but also results in greater memory usage. Option C is the most common real-world configuration for Entity EJBs and is the default setting.

The Activate at setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation are also governed by this setting. Valid values are Once and Transaction. The Once setting indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example when the cache becomes full. The Transaction setting indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is Transaction.

The Load at setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are Activation and Transaction. Activation indicates the bean is loaded when it is activated and implies that the container has exclusive access to the database. Transaction indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is Transaction. The settings of the Activate at and Load at properties govern which commit options are used. For Option A (exclusive database access), use Activate at = Once and Load at = Activation. This option reduces database input/output by avoiding calls to the ejbLoad function, but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.

Note: When using WebSphere Network Deployment and workload management is enabled, Option A cannot be used.

You must use settings that result in the use of Options B or C. For Option B (shared database access), use Activate at = Once and Load at = Transaction. Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to

the ejbActivate function, using Option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over Option A. For Option C (shared database access), use Activate at = Transaction and Load at = Transaction. Load at = Transaction. This option can reduce memory usage by maintaining fewer objects in the cache. However, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

By default, an EJB container runs in the **quick start** mode. The EJB container startup logic delays the loading and processing of all EJB types *except* Message Driven Beans, because message driven beans must exist before messages are posted for them; Startup Beans, which must be processed when the server starts; and EJB types that you specify to initialize when the server starts. .

All other EJB initialization is delayed until the first use of the EJB type. When using local interfaces, the first use is when you perform an InitialContext.lookup method for the type. For remote interfaces, it is when you call the first method on an EJB or its Home.

Enterprise beans back up and recovery best practices

The following items should be considered for backup when using enterprise beans.

Database data

Enterprise beans often use a database for back-end persistence. Container-managed persistence (CMP) entity beans always use a database for back-end persistence. This data should be backed up the same as any of your business data.

The collection for container-managed entity beans persistence is determined by either the schema name specified during deployment, or the schema specified on the data source associated with the enterprise bean. Any persistent store used by session and bean-managed beans is defined by the bean implementation. For database tables, you can choose to save the entire collection or an individual table as shown with the following commands, respectively:

```
SAVLIB LIB(EJB) DEV(*SAVF) SAVF(WSALIB/WSASAVF)
SAVOBJ OBJ(MYBEANTBL) LIB(EJB) DEV(*SAVF) OBJTYPE(*FILE) SAVF(WSALIB/WSASAVF)
```

It might be possible to save database objects while the product is active, if the save operation can obtain a snapshot of the data store. You may have to shut down if a snapshot cannot be obtained. This occurs if there are requests that obtain locks or have open transactions against the database being saved.

Enterprise JavaBeans (EJB) source code, class files, and deployed code

When you deploy enterprise beans, a WebSphere Application Server-specific implementation of the enterprise beans is generated. Save these deployed Java(TM) Archive (JAR) files to avoid redeploying, and to preserve any binding information that was specified during the application installation. The JAR files, application code and configuration, such as bindings, are located by default in the *profile_root*/installedApps directory. By saving this directory, you save your installed applications, including HTML, servlets, JavaServer Pages(TM) (JSP(TM)) files, and enterprise beans. Normally, each application is located in a separate subdirectory, so you can choose to save all applications or a subset.

Note: The commands below have been wrapped for display purposes. Enter each as a single command.

This command saves all installed applications:

```
SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
 OBJ(('/profile_root/installedApps'))
```

This command saves the sampleApp application only:

```
SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
 OBJ(('/profile_root/installedApps/cellName/sampleApp.ear'))
```

If you have located utility or general purpose classes in other directories, such as /*profile_root*/lib/app or /*profile_root*/lib/ext, be sure to include those locations in your backup plan as well.

Administrative configuration

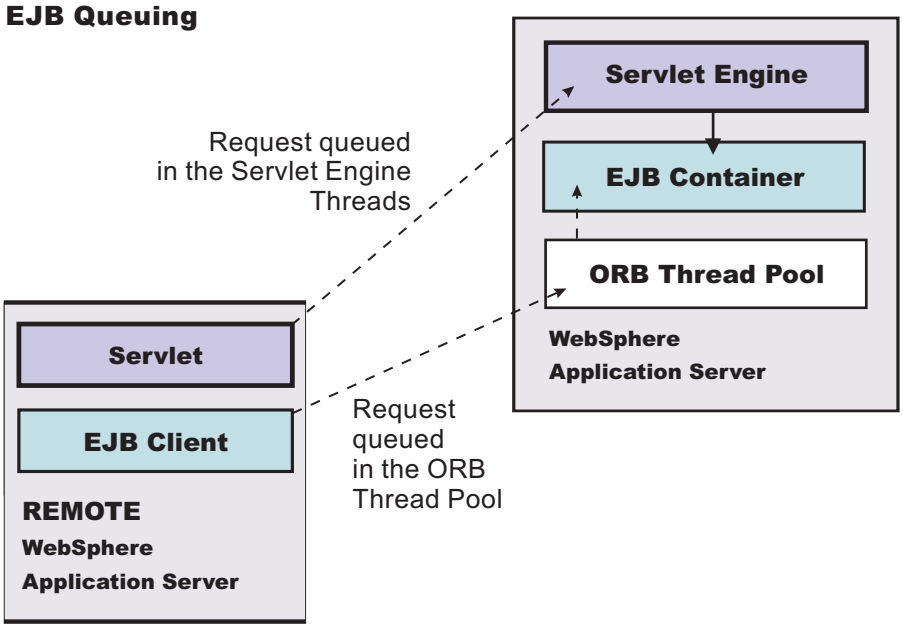
For more information, see Introduction: Administrative configuration data.

EJB method Invocation Queuing

Method invocations to enterprise beans are only queued for remote clients making the method call. An example of a remote client is an Enterprise JavaBeans (EJB) client running in a separate Java virtual machine (JVM) (another address space) from the enterprise bean. In contrast, no queuing occurs if the EJB client, either a servlet or another enterprise bean, is installed in the same JVM on which the EJB method runs, and on the same thread of execution as the EJB client.

Remote enterprise beans communicate by using the Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP). Method invocations initiated over RMI-IIOP are processed by a server-side object request broker (ORB). The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open or closed queue, due to the use of unbounded threads.

The following illustration depicts the two queuing options of enterprise beans.



Enterprise bean and EJB container troubleshooting tips

If you are having problems starting an Enterprise JavaBeans (EJB) container, or encounter error messages or exceptions that appear to be generated on by an EJB container, follow these steps to resolve the problem:

- Use the administrative console to verify that the application server which hosts the container is running.

- Browse the JVM log files for the application server which hosts the container. Look for the message **server *server_name* open for e-business** in the `SystemOut.log` . If it does not appear, or if you see the message **problems occurred during startup**, browse the `SystemErr.log` for details.
- Browse the system log files for the application server which hosts the container.
- Enable tracing for the EJB container component, by using the following trace specification `EJBContainer=all=enabled`. Follow the instructions for dumping and browsing the trace output to narrow the origin of the problem.

If none of these steps solves the problem, check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning. If you do not see a problem that resembles yours, or if the information provided does not solve your problem, contact IBM support for further assistance.

For current information available from IBM Support on known problems and their resolution, see the IBM Support page.

IBM Support has documents that can save you time gathering information needed to resolve this problem. Before opening a PMR, see the IBM Support page.

Application client log error indicates missing JAR file

The following error message appears in the client log file because a Java archive (JAR) file is missing from the classpath on the client machine. The Object Request Broker (ORB) needs this file to unmarshal the nested exception that is part of the EJB exception, returned by the server to the client application. For example, if the EJB returns a DB2® JCC SQL exception nested inside of the EJB exception that it returns to the client, the ORB is not able to unmarshal the nested exception if the db2jcc.jar file that contains the DB2 SQL exception is not in the client classpath.

```
java.rmi.MarshalException: CORBA MARSHAL 0x4942f89a No; nested exception is:
org.omg.CORBA.MARSHAL: Unable to read value
from underlying bridge : Custom marshaling (4) Sender's class does not match
local class  vmcid: 0x4942f000  minor  code: 2202  completed: No*
```

To avoid this error, include the JAR file that contains the class for the nested exception that is returned in the EJB exception.

Enterprise bean cannot be accessed from a servlet, a JSP file, a stand-alone program, or another client

Use these troubleshooting tips for problems related to accessing enterprise beans.

What kind of error are you seeing?

- **javax.naming.NameNotFoundException: Name *name* not found in context ″local″ message** when access is attempted
- **BeanNotReentrantException** is thrown
- **CSITransactionRolledbackException / TransactionRolledbackException** is thrown
- Call fails, Stack trace beginning **EJSContainer E Bean method threw exception [exception_name]** found in JVM log file.
- Call fails, **ObjectNotFoundException or ObjectNotFoundLocalException** when accessing stateful session EJB found in JVM log file.
- Attempt to start container managed persistence (CMP) Enterprise JavaBeans (EJB) module fails with **javax.naming.NameNotFoundException: *dataSourceName***
- **Transaction [tran ID] has timed out after 120 seconds** error accessing EJB.
-
- Symptom: **CNTR0001W: A Stateful SessionBean could not be passivated**
- Symptom: **org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No** returned to client program when attempting to execute an EJB method

If the client is remote to the enterprise bean, which means, running in a different application server or as a stand-alone client, browse the JVM logs of the application server hosting the enterprise bean, as well as log files of the client.

If you do not see a problem that resembles yours, or if the information provided does not solve your problem, perform these steps:

1. If the problem appears to be name-service related, which means that you see a NameNotFoundException error, or a message ID beginning with NMSV, see these topics for more information:
   - Application access problems
   - Naming service troubleshooting tips
2. Check to see if the problem is identified and documented using the links in Diagnosing and fixing problems: Resources for learning.

If you still cannot fix your problem, seeTroubleshooting help from IBM for further assistance.

ObjectNotFoundException or ObjectNotFoundLocalException when accessing stateful session EJB

A possible cause of this problem is that the stateful session bean timed out and was removed by the container. This event must be addressed in the code, according to the EJB 2.1 and later specification. You can review the EJB 2.1 and 3.0 specifications at `http://java.sun.com/products/ejb/docs.html`.

Stack trace beginning ″EJSContainer E Bean method threw exception [exception_name]″ found in JVM log file

If the exception name indicates an exception thrown by an IBM class that begins with ″com.ibm...″, then search for the exception name within the information center, and in the online help as described below. If ″exception name″ indicates an exception thrown by your application, contact the application developer to determine the cause.

javax.naming.NameNotFoundException: Name name not found in context ″local″

A possible reason for this exception is that the enterprise bean is not local (not running in the same Java virtual machine [JVM] or application server) to the client JSP, servlet, Java application, or other enterprise bean, yet the call is to a ″local″ interface method of the enterprise bean . If access worked in a development environment but not when deployed to WebSphere Application Server, for example, it might be that the enterprise bean and its client were in the same JVM in development, but are in separate processes after deployment.

To resolve this problem, contact the developer of the enterprise bean and determine whether the client call is to a method in the local interface for the enterprise bean. If so, have the client code changed to call a remote interface method, or to promote the local method into the remote interface.

References to enterprise beans with local interfaces are bound in a name space local to the server process with the URL scheme of `local:`. To obtain a dump of a server `local:` name space, use the name space dump utility described in the article ″Namespace dump utility for `java:`, `local:` and `server` namespaces.″

BeanNotReentrantException is thrown

This problem can occur because client code, typically a servlet or JSP file, is attempting to call the same stateful SessionBean from two different client threads. This situation often results when an application stores the reference to the stateful session bean in a static variable, uses a global (static) JSP variable to

refer to the stateful SessionBean reference, or stores the stateful SessionBean reference in the HTTP session object. The application then has the client browser issue a new request to the servlet or JSP file before the previous request has completed.

To resolve this problem, ask the developer of the client code to review the code for these conditions.

CSITransactionRolledbackException / TransactionRolledbackException is thrown

An enterprise bean container creates these high-level exceptions to indicate that an enterprise bean call did not complete. When this exception is thrown, browse the JVM logs to determine the underlying cause.

Some possible causes include:
- The enterprise bean might throw an exception that was not declared as part of its method signature. The container is required to roll back the transaction in this case. Common causes of this situation are where the enterprise bean or code that it calls creates a NullPointerException, ArrayIndexOutOfBoundsException, or other Java runtime exception, or where a BMP bean encounters a JDBC error. The resolution is to investigate the enterprise bean code and resolve the underlying exception, or to add the exception to the problem method signature.
- A transaction might attempt to do additional work after being placed in a ″Marked Rollback″, ″RollingBack″, or ″RolledBack″ state. Transactions cannot continue to do work after they are set to one of these states. This situation occurs because the transaction has timed out which, often occurs because of a database deadlock. Work with the application database management tools or administrator to determine whether database transactions called by the enterprise bean are timing out.
- A transaction might fail on commit due to dangling work from local transactions. The local transaction encounters some ″dangling work″ during commit. When a local transactions encounters an ″unresolved action″ the default action is to ″rollback″. You can adjust this action to ″commit″ in an assembly tool. See the assembly tool information center on how to adjust

Attempt to start EJB module fails with ″javax.naming.NameNotFoundException dataSourceName_CMP″exception

This problem can occur because:
- When the DataSource resource was configured, container managed persistence was not selected.
  - To confirm this problem, in the administrative console, browse the properties of the data source given in the NameNotFoundException. On the Configuration panel, look for a check box labeled **Container Managed Persistence**.
  - To correct this problem, select the check box for **Container Managed Persistence**.
- If container managed persistence is selected, it is possible that the CMP DataSource was not bound into the namespace.
  - Look for additional naming warnings or errors in the status bar, and in the hosting application server JVM logs. Check any further naming-exception problems that you find by looking at the topic Application access problems.

Transaction [tran ID] has timed out after 120 seconds accessing an enterprise bean

This error can occur when a client executes a transaction on a CMP or BMP enterprise bean.
- The default timeout value for enterprise bean transactions is 120 seconds. After this time, the transaction times out and the connection closes.
- If the transaction legitimately takes longer than the specified timeout period, go to **Manage Application Servers > *server_name***, select the **Transaction Service properties** page, and look at the property **Total transaction lifetime timeout**. Increase this value if necessary and save the configuration.

Symptom:CNTR0001W: A Stateful SessionBean could not be passivated

This error can occur when a Connection object used in the bean is not closed or nulled out.

To confirm this is the problem, look for an exception stack in the JVM log for the EJB container that hosts the enterprise bean, and looks similar to:

```
[time EDT] <ThreadID> StatefulPassi W CNTR0001W:
A Stateful SessionBean could not be passivated: StatefulBean0
(BeanId(XXX#YYY.jar#ZZZZ, <ThreadID>),
state = PASSIVATING) com.ibm.ejs.container.passivator.StatefulPassivator@<ThreadID>
java.io.NotSerializableException: com.ibm.ws.rsadapter.jdbc.WSJdbcConnection
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at java.io.ObjectOutputStream.outputClassFields((Compiled Code))
 at java.io.ObjectOutputStream.defaultWriteObject((Compiled Code))
 at java.io.ObjectOutputStream.outputObject((Compiled Code))
 at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java(Compiled Code))
 at com.ibm.ejs.container.passivator.StatefulPassivator.passivate((Compiled Code))

 at com.ibm.ejs.container.StatefulBean0.passivate((Compiled Code)
 at com.ibm.ejs.container.activator.StatefulASActivationStrategy.atUnitOfWorkEnd
                     ((Compiled Code))
 at com.ibm.ejs.container.activator.Activator.unitOfWorkEnd((Compiled Code))
 at com.ibm.ejs.container.ContainerAS.afterCompletion((Compiled Code)
```

where *XXX,YYY,ZZZ* is the Bean's name, and *<ThreadID>* is the thread ID for that run.

To correct this problem, the application must close all connections and set the reference to null for all connections. Typically this activity is done in the `ejbPassivate()` method of the bean. Also, note that the bean must have code to reacquire these connections when the bean is reactivated. Otherwise, there are NullPointerExceptions when the application tries to reuse the connections.

Symptom: org.omg.CORBA.BAD_PARAM: Servant is not of the expected type. minor code: 4942F21E completed: No

This error can be returned to a client program when the program attempts to execute an EJB method.

Typically this problem is caused by a mismatch between the interface definition and implementation of the client and server installations, respectively.

Another possible cause is when an application server is set up to use a single class loading scheme. If an application is uninstalled while the application server remains active, the classes of the uninstalled application are still loaded in the application server. If you change the application, redeploy and reinstall it on the application server, the previously loaded classes become back level. The back level classes cause a code version mismatch between the client and the server.

To correct this problem:
1. Change the application server class loading scheme to multiple.
2. Stop and restart the application server and try the operation again.
3. Make sure the client and server code version are the same.

Using access intent policies

You can use access intent policies to help the product runtime environment manage various aspects of Enterprise JavaBeans (EJB) persistence.

About this task

You apply access intent policies to EJB Version 2.0 and 2.1 entity beans, and their methods, by using an assembly tool. A set of default access intent policies comes with the assembly tool. The Rational Application Developer product provides supported assembly tools. Entity beans are not supported in EJB 3.0 modules.

1. Apply default access intent to CMP entity beans. For more information, see the online help available with the assembly tools.
2. Apply access intent policies to methods of container managed persistence entity beans.

Access intent policies

An access intent policy is a named set of properties or access intents that govern data access for Enterprise JavaBeans (EJB) persistence. You can assign policies to an entity bean and to individual methods on an entity bean's home, remote, or local interfaces during assembly. You can set access intents only within EJB Version 2.x-compliant and later modules for entity beans with CMP Version 2.x.

This product supplies a number of access intent policies that specify permutations of read intent and concurrency control; the pessimistic and update policy can be qualified further. The selected policy determines the appropriate isolation level and locking strategy used by the run time environment.

Note: Access intent policies are specifically designed to supplemant the use of isolation level and access intent method-level modifiers found in the extended deployment descriptor for EJB version 1.1 enterprise beans. You cannot specify isolation level and read-only modifiers for EJB version 2.x and later enterprise beans.

Access intent policies configured on an entity basis define the default access intent for that entity. The default access intent controls the entity unless you specify a different access intent policy based on either method-level configuration or application profiling.

Note: Method level access intents were deprecated in Version 6.x.

You can use application profiling or method level access intent policies to control access intent more precisely. Method-level access intent policies are named and defined at the module level. A module can have one or many policies. Policies are assigned, and apply, to individual methods of the declared interfaces of entity beans and their associated home interfaces. A method-based policy is acted upon by the combination of the EJB container and persistence manager when the method causes the entity to load.

For entity beans that are backed by tables with nullable columns, use an optimistic policy with caution. The top down default mapping excludes nullable fields. You can override this when doing a meet-in-the-middle mapping. The fields used in overqualified updates are specified in the ejb-rdb mapping. If nullable columns are selected as overqualified columns, partial update should also be selected.

An entity that is configured with a read-only policy that causes a bean to be activated can cause problems if updates are attempted within the same transaction. Those changes are not committed, and the process displays an exception because data integrity might be compromised.

Concurrency control

Concurrency control is the management of contention for data resources. A concurrency control scheme is considered *pessimistic* when it locks a given resource early in the data-access transaction and does not release it until the transaction is closed. A concurrency control scheme is considered *optimistic* when locks are acquired and released over a very short period of time at the end of a transaction.

The objective of optimistic concurrency is to minimize the time that a given resource is unavailable for use by other transactions. This is especially important with long-running transactions, which under a pessimistic scheme would lock up a resource for unacceptably long periods of time.

Under an optimistic scheme, locks are obtained immediately before a read operation and released immediately after. Update locks are obtained immediately before an update operation and held until the end of the transaction.

To enable optimistic concurrency, this product uses an *overqualified update scheme* to test if the underlying data source has been updated by another transaction since the beginning of the current transaction. With this scheme, the columns marked for update and their original values are added explicitly through a WHERE clause in the UPDATE statement so that the statement fails if the underlying column values have been changed. As a result, this scheme can provide column-level concurrency control; pessimistic schemes can control concurrency at the row level only.

Optimistic schemes typically perform this type of test only at the end of a transaction. If the underlying columns have not been updated since the beginning of the transaction, pending updates to container-managed persistence fields are committed and the locks are released. If locks cannot be acquired or if some other transaction has updated the columns since the beginning of the current transaction, the transaction is rolled back: All work performed within the transaction is lost.

Pessimistic and optimistic concurrency schemes require different transaction isolation levels. Enterprise beans that participate in the same transaction and require different concurrency control schemes cannot operate on the same underlying data connection.

Note: Whether to use optimistic concurrency depends on the type of transaction. Transactions with a high penalty for failure might be better managed with a pessimistic scheme. A high-penalty transaction is one for which recovery is risky or resource-intensive. For low-penalty transactions, it is often worth the risk of failure to gain efficiency through the use of an optimistic scheme. In general, optimistic concurrency is more efficient when update collisions are expected to be infrequent; pessimistic concurrency is more efficient when update collisions are expected to occur often.

Read-ahead hints

Read-ahead schemes enable applications to minimize the number of database round trips by retrieving a working set of container-managed persistence (CMP) beans for the transaction within one query. Read-ahead involves activating the requested CMP beans and caching the data for their related beans, which ensures that data is present for the beans that an application most likely needs next. A *read-ahead hint* is a representation of the related beans to read. The hint is associated with the *findByPrimaryKey* method for the requested bean type, which must be an EJB 2.x-compliant CMP entity bean.

A read-ahead hint takes the form of a character string. You do not have to provide the string; the wizard generates it for you based on the container-managed relationships (CMRs) that are defined for the bean. The following example is provided as supplemental information only. Suppose a CMP bean type A has a finder method that returns instances of bean A. A read-ahead hint for this method is specified using the following notation: `RelB.RelC; RelD`

Interpret the preceding notation as follows:
- Bean type A has a CMR with bean types B and D.
- Bean type B has a CMR with bean type C.

For each bean of type A that is retrieved from the database, its directly-related B and D beans and its indirectly-related C beans are also retrieved. The order of the retrieved bean data columns in each row of the result set is the same as the order in the read-ahead hint: an A bean, a B bean (or null), a C bean (or null), a D bean (or null). For hints in which the same relationship is mentioned more than once, for example, `RelB.RelC;RelB.RelE`, the data columns for a bean occur only once in the result set, at the position the bean first occupies in the hint.

The tokens shown in the notation, like *RelB*, must be CMR field names for the relationships, as defined in the deployment descriptor for the bean. In indirect relationships such as `RelB.RelC`, *RelC* is a CMR field name that is defined in the deployment descriptor for bean type B.

A single read-ahead hint cannot refer to the same bean type in more than one relationship. For example, if a Department bean has an *employees* relationship with the Employee bean and also has a *manager* relationship with the Employee bean, the read-ahead hint cannot specify both *employees* and *manager*.

For more information about how to set read-ahead hints, see the documentation for the Rational Application Developer product.

Run-time behaviors of read-ahead hints

When developing your read-ahead hints, consider the following tips and limitations:

- Read-ahead hints on long or complex paths can result in a query that is too complex to be useful. Read-ahead hints on root or leaf inheritance mappings need particular care. Add up the number of tables that potentially comprise a read-ahead preload to gauge the complexity of the join operations that are required. Consider if the resulting statement constitutes a reasonable query on your target database.
- Read-ahead hints do not work in the following cases:
  - Preload paths across M:N relationships
  - Preload paths across recursive enterprise bean relationships or recursive fk relationships
  - When a read-head hint applies to a SELECT FOR UPDATE statement that requires a table join in a database that does not support the combination of those two operations.

    Generally, the persistence manager issues a SELECT FOR UPDATE statement for a bean only if the bean has an access intent that enforces strict locking policies. Strict locking policies require SELECT FOR UPDATE statements for database select queries. If the database table design requires a join operation to fulfill the statement, many databases issue exceptions because these databases do not support table joins with SELECT FOR UPDATE statements. In those cases, WebSphere Application Server does not implement a read-ahead hint. If the database does provide that support, Application Server implements the read-ahead hints that you configure.

Database deadlocks caused by lock upgrades

To avoid databse deadlocks caused by lock upgrades, you can change the access intent policy for entity beans from the default of wsPessimisticUpdate-WeakestLockAtLoad to wsPessimisticUpdate, or you can use an optimistic locking approach.

When concurrently accessing data, ensure that the application is prepared for database locking that must occur to secure the integrity of the data.

If an entity bean performs a findByPrimaryKey method, which by default obtains a *Read* lock in the database, and the entity bean is updated within the same transaction, a lock upgrade to *Exclusive*.

If this scenario occurs concurrently on multiple threads, a deadlock can happen. This is because multiple read locks can be obtained at the same time but one exclusive lock can only be obtained when the other locks are dropped. Since all transactions are attempting the lock upgrade in this scenario, the one exclusive lock cannot be obtained.

To avoid this problem, you can change the access intent policy for the entity bean from the default of wsPessimisticUpdate-WeakestLockAtLoad method to wsPessimisticUpdate method. This change enables the application to inform the product and the database that the transaction has updated the enterprise bean. The *Update* lock is immediately obtained on the findByPrimaryKey method. This avoids the lock upgrade when the update is performed at a later time.

The preferred technique to define access intent policies is to change the access intent for the entire entity bean. You can change the access intent for the findByPrimaryKey method, but this was deprecated in Version 6. You might want to change the access intent for an individual method if, for example, the entity bean is involved in some transactions that are read only.

An alternative technique is to use an optimistic approach, where the findByPrimaryKey method does not hold a read lock, so there is no lock upgrade. However, this requires that the application is coded for this in order to handle rollbacks. Optimistic locking is intended for applications that do not expect database contention on a regular basis.

To change the access intent policy for an entity bean, you can use the assembly tool to set the bean level, as described in “Applying access intent policies to beans” on page 165.

Access intent assembly settings

Access intent policies contain data-access settings for use by the persistence manager. Default access intent policies are configured on the entity bean.

These settings are applicable only for EJB 2.x and EJB 3.0-compliant entity beans that are packaged in EJB 2.x and EJB 3.0-compliant modules. Connection sharing between beans with bean-managed persistence and those with container-managed persistence is possible if they all use the same access intent policy.

Name:

Specifies a name for a mapping between an access intent policy and one or more methods.

Description:

Contains text that describes the mapping.

Methods - name:

Specifies the name of an enterprise bean method, or the asterisk character (*). The asterisk is used to denote all of the methods of an enterprise bean’s remote and home interfaces.

Methods - enterprise bean:

Specifies which enterprise bean contains the methods indicated in the Name setting.

Methods - type:

Used to distinguish between a method with the same signature that is defined in both the home and remote interface. Use `Unspecified` if an access intent policy applies to all methods of the bean.

Data type	String
Range	Valid values are `Home`, `Remote`,`Local`, `LocalHome` or `Unspecified`

Methods - parameters:

Contains a list of fully qualified Java type names of the method parameters. This setting is used to identify a single method among multiple methods with an overloaded method name.

Applied access intent:

Specifies how the container must manage data access for persistence. Configurable both as a default access intent for an entity and as part of a method-level access intent policy.

Data type	String
Default	`wsPessimisticUpdate-WeakestLockAtLoad`. With Oracle, this is the same as wsPessimisticUpdate.

Range

Valid settings are `wsPessimisticUpdate`, `wsPessimisticUpdate-NoCollision`, `wsPessimisticUpdate-Exclusive`, `wsPessimisticUpdate-WeakestLockAtLoad`, `wsPessimisticRead`, `wsOptimisticUpdate`, or `wsOptimisticRead`. Only `wsPessimisticRead` and `wsOptimisticRead` are valid when class-level caching is enabled in the EJB container.

This product supports lazy collections. For each segment of a collection, iterating through the collection (*next()*) does not trigger a remote method call to retrieve the next remote reference. Two policies (`wsPessimisticUpdate` and `wsPessimisticUpdate-Exclusive`) are extremely lazy; the collection increment size is set to 1 to avoid overlocking the application. The other policies have a collection increment size of 25.

If an entity is not configured with an access intent policy, the run-time environment typically uses `wsPessimisticUpdate-WeakestLockAtLoad` by default. If, however, the **Lifetime in cache** property is set on the bean, the default value of **Applied access intent** is `wsOptimisticRead`; updates are not permitted.

Additional information about valid settings follows:

Profile name	Concurrency control	Access type	Transaction isolation
wsPessimisticRead (Note 1)	pessimistic	read	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate (Note 2)	pessimistic	update	For Oracle, read committed. Otherwise, repeatable read
wsPessimisticUpdate-Exclusive (Note 3)	pessimistic	update	serializable
wsPessimisticUpdate-NoCollision (Note 4)	pessimistic	update	read committed
wsPessimisticUpdate-WeakestLockAtLoad (Note 5)	pessimistic	update	Repeatable read
wsOptimisticRead	optimistic	read	read committed
wsOptimisticUpdate (Note 6)	optimistic	update	read committed

Notes:

1. Read locks are held for the duration of the transaction.
2. The generated SELECT FOR UPDATE query grabs locks at the beginning of the transaction.
3. SELECT FOR UPDATE is generated; locks are held for the duration of the transaction.
4. A plain SELECT query is generated. No locks are held, but updates are permitted. Use cautiously. This intent enables execution without concurrency control.
5. Where supported by the backend, the generated SELECT query does not include `FOR UPDATE`; locks are escalated by the persistent store at storage time if updates were made. Otherwise, the same as wsPessimisticUpdate.
6. Generated overqualified-update query forces failure if CMP column values have changed since the beginning of the transaction.

   Be sure to review the rules for forming overqualified-update query predicates. Certain column types (for example, BLOB) are ineligible for inclusion in the overqualified-update query predicate and might affect your design.

Applying access intent policies to beans

You can apply an access intent policy to an application's entity beans through the assembly tool.

About this task

Container-managed persistence (CMP) developers can use *access intent* to provide hints on how the application server run time should manage the details of persistence without having to explicitly manage any of the persistence logic from within their application.

Using the access intent service is also an option for programmers who develop bean-managed persistence (BMP) entity beans. Because the only meaningful difference between BMP and CMP components is the mechanism that provides the persistence logic, BMP beans leverage access intent hints in the same manner as the EJB container manages accent intent for CMP beans. This ability becomes especially important when BMP entities and CMP entities want to share connections. BMP beans configured with the same concurrency as the CMP beans and implemented to the same isolation level mapping as the CMP can share connections.

Developers can apply access intent policies to BMP entity beans as well as to CMP entity beans. It is expected that BMP developers use only those access intent attributes that are important to a particular BMP bean. The access intent service interface is bound into the *java:comp namespace* for each particular BMP bean. The access intent policy retrieved from the access intent service is current from the time that the *ejbLoad* process is called until the time that the *ejbStore* process completes its invocation.

Note: This is the preferred technique to define access intent policies. Method-level access intent is deprecated in Version 6.0.

1. Start an assembly tool.
2. Optional: Open the Java EE perspective to work with Java EE projects. Click **Window > Open Perspective > Other > Java EE**.
3. Optional: Open the Project Explorer view. Click **Window > Show View > Project Explorer**. Another helpful view is the Navigator view (**Window > Show View > Navigator**).
4. Create a new application EAR file or edit an existing one.

   For example, to change attributes of an existing application, use the import wizard to import an EAR file. To start the import wizard:

   a. Select **File > Import > EAR file > Next**
   b. Select the EAR file.
   c. Create a WebSphere Application Server v6.0 type of Server Runtime. Select **New** to open the New Server Runtime Wizard and follow the instructions.
   d. In the *Target server* field, select *WebSphere Application Server v6.0* type of Server Runtime.
   e. Select **Finish**
5. In the Project Explorer view of the J2EE perspective, right-click **Deployment Descriptor: EJB Module Name** under the EJB module for the bean instance, then select **Open With > Deployment Descriptor Editor**. A property dialog notebook for the EJB project is displayed in the property pane.
6. Select the **Access** tab.
7. In the **Access Intent for Entities 2.x (Bean Level)** panel, select the name of the bean.
8. On the right side of the **Access Intent for Entities 2.x (Method Level)** panel, select **Add**. The **Add Access Intent** panel displays.
9. In the **Access intent name** field, select wsPessimisticUpdate from the drop-down list.
10. Optional: Enter a **Description** to help you remember what this policy does.
11. Optional: Change the **Persistence Option** setting
12. Click **Finish**. The access intent policy for the entity bean is shown in the **Access Intent for Entities 2.x (Bean Level)** panel

Configuring read-read consistency checking with an assembly tool

Read-read consistency checking only applies to LifeTimeInCache beans whose data is read from another transaction. For the Access Intents that are for *repeatable read* (RR), this means the product checks that the data is consistent with that in the data store, and ensures that no one updates it after the checking.

About this task

For the access intents that are for *read committed* (RC), this means the product checks that the data is consistent at the point of checking, it does **not** guarantee that the data does not change after the checking. This makes the behavior of the LifeTimeInCache bean the same as non-LifeTimeInCache beans.

To perform this checking, you need to configure CMP entity beans with read-read consistency checking. You can do this using an assembly tool. To learn how to complete this task see the topic, ″Adding bean-level access intent for entity beans 2.x″ in the assembly tool documentation at http://publib.boulder.ibm.com/infocenter/radhelp/v7r5mbeta/topic/com.ibm.jee5.doc/topics/cejb3.html

Example: Read-read consistency checking

Read-read consistency checking only applies to LifeTimeInCache beans whose data is read from another transaction.

Usage scenario

For the access intents that are for *repeatable read* (RR), this means the product checks that the data is consistent with that in the data store and ensures that no one updates it after the checking. For the Access Intents that are for *read committed* (RC), this means the product checks that the data is consistent at the point of checking, but it does **not** guarantee that the data does not change after the checking. This makes the behavior of the LifeTimeInCache bean the same as non-LifeTimeInCache beans.

You have three options for setting consistency checking, as shown in the following scenarios concerning the calculation of interest in ″Ann's″ bank account. In each case, the data store is shared by this Enterprise JavaBeans (EJB) container managed persistence (CMP) application to calculate the interest and other applications, such as EJB bean managed persistence (BMP) , Java Database Connectivity (JDBC), or legacy applications. Also in each case, the EJB account is configured as a *long-lifetime* bean.

NONE

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from cache, with a balance of $100.
- Calculate Ann's interest, but the result might not be correct because of the data integrity issue.

Read-read checking AT_TRAN_BEGIN

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from cache, with a balance of $100.

- PM performs read-read check on Ann's account and finds that the balance of 100 is changed. It issues a database query to retrieve balance of $120, and Ann's data in the cache is refreshed.
- Calculate Ann's interest, proceed with the transaction because data integrity is protected.

Read-read checking AT_TRAN_END

- The server is started.
- User 1 in Transaction 1 calls Account.findByPrimaryKey(″10001″), account data for Ann is read from the database, with a balance of $100.
- Ann's record is cached by the persistence manager (PM) on the server.
- User 2 writes a JDBC call and changes the balance to $120.
- User 3 in Transaction 2 calls Account.findByPrimaryKey() for account ″10001″, Ann's data is read from database, with balance of $100.
- Calculate Ann's interest.
- During end of transaction 2, PM performs read-read check on Ann's account and finds that the balance of 100 is changed.
- PM rolls back the transaction and invalidates the cache. The transaction fails and again data integrity is protected.

Access intent service

Access intent is a WebSphere Application Server runtime service that enables you to precisely manage an application's persistence.

The access intent service defines a set of declarative annotations used by the Enterprise JavaBeans (EJB) container and its agents to make performance optimizations for entity bean access. These annotations are organized into sets called *access intent policies*.

Access intent policies contain a set of annotations considered as hints by the EJB container and its agents. Most access intent policies are hints representing high-level abstractions that can be mapped to a specific back end resource manager. It is the responsibility of the EJB persistence machinery to ensure the necessary concurrency control, connection, and cache management when carrying out the persistence details. The EJB persistence manager can use access intent hints to make better performance decisions when carrying out its assigned task. A smaller number of access intents are hints to the EJB container, influencing the management of EJB collections.

Typically, you configure *bean level* access intent for your applications. You can also apply access intent policies to beans within the scope of *application profiles*. Consequently, you can configure beans with multiple and opposing access intent policies. The application profiling documentation explains in more detail how to configure an application to apply a particular access intent policy to a bean for one request, then apply another access intent policy to the same bean for a different request.

Support for applying access intent policies at the method level is deprecated in WebSphere Application Server Version 6.0. In this practice of configuring access intent, you apply a policy to methods within the scope of an EJB module so that the policy becomes the default access intent for all requests upon those methods.

Access intent design considerations

Note: Refrain from over-tuning an application. You can introduce errors by incorrectly using the access intent service. For example, misuse of the wsPessimisticUpdate-NoCollision policy can result in lost updates; inappropriately setting the collection increment value can introduce performance issues; and problem determination is more difficult when an application is configured with multiple access intent policies.

Note: Clarity and simplicity should be your guiding principles when using the access intent service. This is even more important when applying access intent polices within the scope of application profiles.

Even though access intent policies can be configured on any method of an entity bean, some attributes of a policy can only be leveraged by the runtime environment under certain conditions. For example, concurrency and access intent are only used for CMP entity beans when the ejbLoad method is driven to open a connection and read data from a given resource; that data is cached and used to drive the proper queries during invocation of the ejbStore method. Read-ahead hints are only used during the execution of a finder for a bean. The collection increment and resource manager prefetch increment are only used on multi-object finders. Configuring policies on methods that do not use the policy is not an error. Only certain attributes of any policy are used, even when the policy is appropriately applied to a method. However, configuring policies unnecessarily throughout an application obscures the design of the application and complicates the maintenance of the application.

Access intent with BMP entity beans

Access intent's declarative functionality provides great power to you as a CMP entity bean developer. You can provide hints on how the product should manage the details of persistence without having to explicitly manage any of the persistence logic in the application. There are situations, however, in which you might need to develop BMP entity beans. Since the only meaningful difference between BMP and CMP components is who provides the persistence logic, BMP entity beans should be able to leverage access intent hints just as the product does on behalf of CMP entity beans. BMP entity beans that use the access intent service participate in application profiling; that is, the value of the access intent attributes can differ from request to request, allowing the BMP entity bean to seamlessly modify its persistence strategy.

You can apply access intent policies to BMP entity bean methods as well as CMP entity bean methods. Because access intent hints are not contractual in nature, there is no obligation for a BMP entity bean to exploit them. BMP entity beans are expected to use only those access intent attributes that are important to that particular bean.

The current access intent policy is bound into the java:comp namespace for a particular BMP entity bean. That policy is current only for the duration of the method call during which the access intent policy was retrieved. In a typical scenario, you would cache the access type during invocation of the ejbLoad method so that appropriate actions can be taken during invocation of the ejbStore method.

Access intent best practices

When applying access intent policies to EJB methods, consider the following issues.

- **Start by configuring the default access intent policy for an entity.** After your application is built and started, you can tune certain access paths in your application using application profiling or method-level access intent.
- **Don't mix access types.** Avoid using both pessimistic and optimistic policies in the same transaction. For most databases, pessimistic and optimistic policies use different isolation levels. This can result in multiple database connections, which prevents you from taking advantage of the performance benefits possible through connection sharing.
- **Take care when applying the `wsPessimisticUpdate-NoCollision` policy.** This policy does not ensure data integrity. No database locks are held, so concurrent transactions can overwrite each other's updates. Use this policy only if you can be sure that only one transaction attempts to update persistent store at any time.

For further information on Java Persistence API (JPA) Access intent, see the topic on JPA Access intent.

Applying access intent policies to methods

You apply an access intent policy to a method, or set of methods, in an application's entity beans through the assembly tool.

About this task

Note: Method-level access intent is deprecated in Version 6.0.

1. Start an assembly tool.
2. Optional: Open the Java EE perspective to work with Java EE projects. Click **Window > Open Perspective > Other > Java EE**.
3. Optional: Open the Project Explorer view. Click **Window > Show View > Project Explorer**. Another helpful view is the Navigator view (**Window > Show View > Navigator**).
4. Create a new application EAR file or edit an existing one.

   For example, to change attributes of an existing application, use the import wizard to import an EAR file. To start the import wizard:

   a. Select **File > Import > EAR file > Next**
   b. Select the EAR file.
   c. Create a WebSphere Application Server v6.0 type of Server Runtime. Select **New** to open the New Server Runtime Wizard and follow the instructions.
   d. In the *Target server* field, select *WebSphere Application Server v6.0* type of Server Runtime.
   e. Select **Finish**
5. In the Project Explorer view of the J2EE perspective, right-click the **Deployment Descriptor: EJB Module Name** under the EJB module for the bean instance, then select **Open With > Deployment Descriptor Editor**. A property dialog notebook for the EJB project is displayed in the property pane.
6. Select the **Access** tab.
7. On the right side of the **Access Intent for Entities 2.x (Method Level)** panel, select **Add**. The **Add Access Intent** panel displays.
8. Specify the **Name** for your new intent policy.
9. Select the **Access intent name** from the drop-down list.
10. Enter a **Description** to help you remember what this policy does.
11. Optional: Select **Read Ahead Hint**. A single access intent read ahead hint might not refer to the same bean type in more than one relationship. For example, if a **Department** enterprise bean has a relationship *employees* with the **Employee** enterprise bean, and also has a relationship *manager* with the **Employee** enterprise bean, then a read ahead hint cannot specify both *employees* and *manager*.
12. Click **Next**. The next **Add Access Intent** panel displays, with optional attributes.
13. Optional: Decide whether or not to overwrite these optional access intent attributes. Click on those you want to change.
14. Click **Next**. The next **Add Access Intent** panel, with a list of Enterprise Beans, displays.
15. Select one or more Enterprise Beans from the list.

    Note: If you selected **Read Ahead Hint** in an earlier step, you can only select **ONE** bean at this step.
16. Click **Next**. The next **Add Access Intent** panel, with a list of methods, displays.
17. Select the methods you want to use.
18. If you *DID NOT* select **Read Ahead Hint** in an earlier step, click **Finish**. If you *DID* select the Read Ahead Hint option, you can click **Next** to specify your Read Ahead Hint for the specified bean. The next **Add Access Intent** panel, with a list of EJB preload paths, displays.
19. Edit the EJB preload path by selecting relationship roles from the **Relationship roles:** window.
20. Click **Finish**. A new entry is created in the **Access Intent for Entities 2.x (Method Level)** panel

Using the AccessIntent API

This task describes how to programmatically retrieve and call the AccessIntent API during the execution of bean managed persistence (BMP) entity bean methods.

1. Look up the access intent service from the namespace. For example:

```
InitialContext ic = new InitialContext();
AccessIntentService aiService = ic.lookup("java:comp/websphere/AppProfile/AccessIntentService");
```

2. From a method of the remote or local component interface of the BMP, get the current AccessIntent object using the javax.ejb.EntityContext. This is passed to the BMP when the container calls the setEntityContext method. Assume the EntityContext was stored in a variable named myEntityCtx. For example:

```
AccessIntent ai = aiService.getAccessIntent (myEntityCtx);
```

3. Use the get() methods of AccessIntent interface to obtain the desired information. For example:

```
int concurrency = ai.getConcurrencyControl();
int accessType = ai.getAccessType();
if ( (concurrency == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC)
   && (accessType == AccessIntent.ACCESS_TYPE_UPDATE) ) {
     int exclusive = ai.getPessimisticUpdateLockHint();
     // . . .
}
// . . .
```

   For a detailed example of the use of the AccessIntent API, see Example: Using IBM extended APIs to share connections between CMP beans and BMP beans.

Results

Note: The access intent object reference retrieved from the java:comp lookup is current for the duration of the method in which the reference was looked up. Depending on how you configured the application profile, subsequent calls of the same method might not retrieve the same access intent reference. You can only look up the object reference during the call of a BMP entity bean's method; the reference does not exist during a request on a container managed persistence (CMP) entity bean. Therefore, access intent object references should not be cached beyond, or used outside of, the scope of the execution of any given BMP method.

AccessIntent interface

The AccessIntent interface is available to bean-managed persistence (BMP) entity beans.

A BMP entity bean can get and use an instance of the AccessIntent interface. For more information see "Using the AccessIntent API" on page 170.

AccessIntent interface

```
package com.ibm.websphere.appprofile.accessintent;

/**
* This interface defines the essential access intents
* available at run time.
*/
public interface AccessIntent {

/**
* Returns the concurrency control intent, which indicates
* the application prefers either pessimistic or optimistic
* concurrency control when accessing the current component
* in the context of the current transaction.
*/
public int getConcurrencyControl();
public final int CONCURRENCY_CONTROL_PESSIMISTIC = 1;
public final int CONCURRENCY_CONTROL_OPTIMISTIC = 2;

/**
* Returns access type intent, which indicates the application
* intends either update or read access of the current component
* in the context of the current transaction.
```

```
*/
public int getAccessType();
public final int ACCESS_TYPE_UPDATE= 1;
public final int ACCESS_TYPE_READ = 2;

/**
* Returns an integer value that indicates that the run time should
* assume that there will be no collision on retrieved rows.
*/
public int getPessimisticUpdateLockHint();
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;

/*
* Returns an integer value that indicates that the run time should
* assume that there will be collisions on retrieved rows.
*/
public int getPessimisticUpdateLockHint();
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NOCOLLISION = 1;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_WEAKEST_LOCK_AT_LOAD = 2;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_NONE = 3;
public final static int PESSIMISTIC_UPDATE_LOCK_HINT_EXCLUSIVE = 4;


/**
* Returns the collection access intent, which indicates the
* application intends to access the objects returned by the
* currently executing finder in either serial or random fashion.
*/
public int getCollectionAccess();
public final int COLLECTION_ACCESS_RANDOM = 1;
public final int COLLECTION_ACCESS_SERIAL = 2;

/**
* Returns the collection scope, which indicates the maximum
* lifespan of a lazy collection.
*/
public int getCollectionScope();
public final int COLLECTION_SCOPE_TRANSACTION = 1;
public final int COLLECTION_SCOPE_ACTIVITYSESSION = 2;
public final int COLLECTION_SCOPE_TIMEOUT = 3;

/**
* Returns the timeout value in seconds when collectionScope is Timeout.
*/
public int getCollectionTimeout();

/**
* Returns the number of elements the application requests be contained
* in each segment of the element collection returned by the currently
* executing finder.
*/
public int getCollectionIncrement();

/**
* Returns the ReadAheadHint requested by the application for the currently
* executing finder.
*/
public ReadAheadHint getReadAheadHint();

/**
* Returns the number of elements the application requests be contained in
* each segment of a query made on a database.
```

```
*/
public int getResourceManagerPreFetchIncrement();

}
```

Access intent exceptions

The exceptions thrown in response to the application of access intent policies are listed.

com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException

If the method that drives the ejbLoad() method is configured to be read-only but updates are then made within the transaction that loaded the bean's state, an exception is thrown during invocation of the ejbStore() method, and the transaction is rolled back. Likewise, the ejbRemove() method cannot succeed in a transaction that is set as read-only. If an update hint is applied to methods of entity beans with bean-managed persistence, the same behavior and exception results. The forwarded exception object contains the message string `PMGR1103E: update instance level read only bean` *beanName*

This exception is also thrown if the applied access intent policy cannot be honored because a finder, ejbSelect, or container-managed relationship (CMR) accessor method returns an inherently read-only result. The forwarded exception object contains the message string `PMGR1001: No such DataAccessSpec -` *methodName*

The most common occurrence of this error is when a custom finder that contains a read-only EJB Query Language (EJB QL) statement is called with an applied access intent of `wsPessimisticUpdate` or `wsPessimisticUpdate-Exclusive`. These policies require the use of a USE AND KEEP UPDATE LOCKS clause on the SQL SELECT statement to be executed, but a read-only query cannot support USE AND KEEP UPDATE LOCKS. Other examples of read-only queries include joins; the use of ORDER BY, GROUP BY, and DISTINCT keywords.

To eliminate the exception, edit the EJB query so that it does not return an inherently read-only result or change the access intent policy being applied.

- If an update access is required, change the applied access intent setting to `wsPessimisticUpdate-WeakestLockAtLoad` or `wsOptimisticUpdate`.
- If update access is not truly required, use `wsPessimisticRead` or `wsOptimisticRead`.
- If connection sharing between entity beans is required, use `wsPessimisticUpdate-WeakestLockAtLoad` or `wsPessimisticRead`.

com.ibm.websphere.ejb.container.CollectionCannotBeFurtherAccessed

If a lazy collection is driven after it is no longer in scope, and beyond what has already been locally buffered, a CollectionCannotBeFurtherAccessed exception is thrown.

com.ibm.ws.exception.RuntimeWarning

If an application is configured incorrectly, a run-time warning exception is thrown as the application starts; startup is ended. You can validate an application's configuration by choosing the verify function. Some examples of misconfiguration include:

- A method configured with two different access intent policies
- A method configured with an undefined access intent policy

Access intent troubleshooting tips

The following frequently asked questions involving access intent are answered.

Oracle database fails when no access policies are applied

No access intent policies have been applied and the application runs with a DB2 database, but it fails with an Oracle database with the following message:

```
com.ibm.ws.ejbpersistence.utilpm.PersistenceManagerException: PMGR1001E: No such DataAccessSpec
:FindAllCustomers. The backend datastore does not support the SQLStatement needed by this
AccessIntent: (pessimistic update-weakestLockAtLoad)(collections: transaction/25) (resource
manager prefetch: 0) (AccessIntentImpl@d23690a).
```

If you have not configured access intent, all of your data is accessed under the default access intent policy (`wsPessimisticUpdate-WeakestLockAtLoad`). On DB2 the weakest lock is share. On Oracle databases, however, the weakest lock is update; this means that the SQL query must contain a FOR UPDATE clause. To avoid this problem, try to apply an access intent policy that supports optimistic concurrency.

Calling a finder method displays an InconsistentAccessIntentException at run time

This can occur when you use method-level access intent policies to apply more control over how a bean instance is loaded. This exception indicates that the entity bean was previously loaded in the same transaction. This could happen if you called a multifinder method that returned the bean instance with access intent policy X applied; you are now trying to load the second bean again by calling its findByPrimaryKey method with access intent Y applied. Both methods must have the same access intent policy applied.

Likewise, if the entity was loaded once in the transaction using an access intent policy configured on a finder, you might have called a container-managed relationship (CMR) accessor method that returned the entity bean configured to load using that entity's default access intent.

To avoid this problem, ensure that your code does not load the same bean instance twice within the same transaction with different access intent policies applied. Avoid the use of method-level access intent unless absolutely necessary.

An InconsistentAccessIntentException displays in a container-managed relationship with two beans

There are two beans in a container-managed relationship. The findByPrimaryKey method is called on the first bean and then the getBean2 method is called and a CMR accessor method is called, on the returned instance and an InconsistentAccessIntentException displays.

You are probably using read-ahead. When you loaded the first bean, you caused the second bean to be loaded under the access intent policy applied to the finder method for the first bean. However, you have configured your CMR accessor method from the first bean to the second with a different access intent policy. CMR accessor methods are really finder methods in disguise; the run-time environment behaves as if you were trying to change the access intent for an instance you have already read from persistent store.

To avoid this problem, beans configured in a read-ahead hint are all driven to load with the same access intent policy as the bean to which the read-ahead hint is applied.

A bean with a one-to-many relationship to a second bean displays an UpdateCannotProceedWithIntegrityException error when an instance of the second bean is added to the first bean's collection

A bean with a one-to-many relationship to a second bean displays an UpdateCannotProceedWithIntegrityException error when an instance of the second bean is added to the first bean's collection. The first bean has a pessimistic-update intent policy applied.

The second bean probably has a read intent policy applied. When you add the second bean to the first bean's collection, you are not updating the first bean's state, you are implicitly modifying the second bean's state. The second bean contains a foreign key to the first bean, which is modified.

To avoid this problem, ensure that both ends of the relationship have an update intent policy applied if you expect to change the relationship at run time.

Managing EJB containers

Each application server can have a single Enterprise JavaBeans (EJB) container; one is created automatically for you when the application server is created. The following steps are to be performed only as needed to improve performance after the EJB application has been deployed.

1. Adjust EJB container settings.
2. Adjust EJB cache settings.

What to do next

If adjustments do not improve performance, consider adjusting access intent policies for entity beans, reassembling the module, and redeploying the module in the application.

EJB container settings

Use this page to configure and manage the EJB container of this application server.

To view this administrative console page, click **Servers** → **WebSphere application servers** → **server** → **EJB Container Settings** → **EJB container**.

Passivation directory

Specifies the directory into which the container saves the persistent state of passivated stateful session beans. This directory must already exist. It is not automatically created.

Stateful session beans with an activation policy of TRANSACTION are passivated at the end of the transaction in which they are enlisted, and stateful session beans with an activation policy of ONCE (default) are passivated when the number of active bean instances becomes greater than the cache size specified in the container configuration. When a stateful bean is passivated, the container serializes the bean instance to a file in the passivation directory and discards the instance from the bean cache. If, at a later time, a request arrives for the passivated bean instance, the container retrieves it from the passivation directory, deserializes it, returns it to the cache, and dispatches the request to it. If any step fails (for example, if the bean instance is no longer in the passivation directory), the method invocation fails.

Inactive pool cleanup interval

Specifies the interval at which the container examines the pools of available bean instances to determine if some instances can be deleted to reduce memory usage. This setting is for all bean pools. **Attention:** Stateful session beans are NOT pooled, so this applies to stateless session and entity bean pools.

Data type	Integer
Default	30000
Units	Milliseconds
Range	0 to 2 147 483 647

Default data source JNDI name

Specifies the JNDI name of a data source to use if no data source is specified during application deployment. This setting is not applicable for EJB 2.x-compliant CMP beans.

Servlets and enterprise beans use *data sources* to obtain these connections. When configuring a container, you can specify a default data source for the container. This data source becomes the default data source used by any entity beans installed in the container that use container-managed persistence (CMP).

The default data source for a container is secure. When specifying it, you must provide a user ID and password for accessing the data source.

Specifying a default data source is optional if each CMP entity bean in the container has a data source specified in its configuration. If a default data source is not specified and a CMP entity bean is installed in the container without specifying a data source for that bean, applications cannot use that CMP entity bean.

Enable stateful session bean failover using memory-to-memory replication

Specifies that failover is enabled for *all* stateful session beans installed in this EJB container.

This checkbox is disabled until you define a replication domain. This selection has a hyperlink to help you configure the replication settings. If no replication domains are configured, the link takes you to a panel where you can create one. If at least one domain is configured, the link takes you to a panel where you can select the replication settings to be used by the EJB container.

Data type	Checkbox
Default	Unselected
Range	Selected or unselected.

Initial state

Specifies the execution state requested when the server first starts.

Data type	String
Default	Started
Range	Valid values are `Started` and `Stopped`

EJB container system properties

In addition to the settings accessible from the administrative console, you can set the following system property by command-line scripting.

com.ibm.websphere.ejbcontainer.poolSize

Specifies the size of the pool for the specified bean type. This property applies to stateless, message-driven and entity beans. If you do not specify a default value, the container defaults of 50 and 500 are used.

Set the pool size for a given entity bean as follows:

```
beantype= [H]
min,[H]
max [:beantype=
[H]min,
[H]max...]
```

beantype is the Java EE name of the bean, formed by concatenating the application name, the **#** character, the module name, the **#** character, and the name of the bean, that is, the string assigned to the `<ejb-name>` field in the bean's deployment descriptor. *min* and *max* are the minimum and maximum pool sizes, respectively, for that bean type. Do not specify the square brackets shown in the previous prototype; they denote optional additional bean types that you can specify after the first. Each bean-type specification is delimited by a colon (:).

Use an asterisk (*) as the value of *beantype* to indicate that all bean types are to use those values unless overridden by an exact bean-type specification somewhere else in the string, as follows:

```
*=30,100
```

To specify that a default value be used, omit either *min* or *max* but retain the comma (,) between the two values, as follows (split for publication):

```
SMApp#PerfModule#TunerBean=54,
   :SMApp#SMModule#TypeBean=100,200
```

You can specify the bean types in any order within the string.

You can designate the maximum configured EJB pool size as a *hard limit* by inserting the character *H* directly in front of the *max* value. Without the *H*, the maximum value indicates how many EJB instances can be pooled, and does not limit the number of EJB instances that can be created or in use. Inserting the *H* before the max value indicates that it is a hard limit, and the EJB Container blocks creation of more instances when that limit is reached. Further threads must wait until an instance becomes available or until the transaction times out.

You can designate the minimum configured EJB pool size as a *hard limit* by inserting the character *H* directly in front of the *min* value. Without the *H*, the minimum value indicates how many EJB instances are maintained in the pool when the EJB type is not actively in use, but does not pre-load the pool when the application is started. Normally, the minimum pool size is not reached until the minimum number of EJB instances has been accessed concurrently by the application. Inserting the *H* before the min value indicates that it is a hard limit, and the EJB Container pre-loads the pool with the minimum number of EJB instances when the application is started.

As an example, if you want to indicate that no more than 200 EJB instances are created, and that additional requests must wait for an instance and might time out while waiting, you enter:

```
SMApp#SMModule#TypeBean=100,H200
```

If you want to indicate that the EJB Container pre-load the pool with a minimum of 100 EJB instances when the application is started, you enter:

```
SMApp#SMModule#TypeBean=H100,200
```

Note: The hard limit indicator is only available to EJB Version 2.0 or later Stateless Session beans.

com.ibm.websphere.ejbcontainer.allowEarlyInsert

This property is applicable to CMP 1.1 beans only. By default, the EJB Container creates the entity bean representation in the database only after the method ejbPostCreate(...) is called.

Note: CMP beans are not supported in EJB 3.0 modules.

Some applications may rely on method ejbCreate(...) to have created the entity bean in the database. For such a requirement, setting the JVM property *com.ibm.websphere.ejbcontainer.allowEarlyInsert* to **true** overrides the default behavior.

Changing enterprise bean types to initialize at application start time using the administrative console

All Enterprise JavaBeans (EJB) types within a server can be forced to initialize at application start time by setting a system property within the administrative console.

About this task

If the value of this property is set to **true**, then all beans within the server are initialized at each application's start time.

However, by default, the product's EJB container delays the initialization (loading of classes and processing of deployment descriptor metadata) of most EJB types until they are needed during run time. This delay helps to speed up the application start time.

1. Open the administrative console.
2. Select **Servers**.
3. Select **Application Servers**.
4. Select the server you want to configure.

5. In the Server Infrastructure area, select **Java and Process Management**.
6. In the Server Infrastructure area, select **Process Definition**.
7. In the Additional Properties area, select **Java Virtual Machine**.
8. In the Additional Properties area, select **Custom Properties**.
9. Select the **New** box.
10. In the **Name** entry field, type *com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup*.
11. In the **Value** entry field, type *true*. Entering *true* causes all Enterprise JavaBeans to initialize when your application starts. Entering *false* causes initialization of all beans to be delayed.

    Note: Setting com.ibm.websphere.ejbcontainer.initializeEJBsAtStartup to either true or false takes precedence over any *Start EJB at Application Start* settings made on individual EJB types.
12. Select **OK**.

What to do next

This task can also be done by using an assembly tool.

Tuning the EJB cache using the trace service

About this task

The size of your Enterprise JavaBeans (EJB) cache can effect the performance of the application server. One of the steps in tuning your EJB container to optimum performance levels is to fine tune the EJB cache. The following describes how to use the diagnostic trace service to help you determine the best cache size for you.

1. Enable the EJB cache trace. To learn about working with the trace service, see Working with trace. For information about the trace service settings, see Diagnostic trace service settings.

   Set up your trace to use this trace string:

```
com.ibm.ejs.util.cache.BackgroundLruEvictionStrategy=all=enabled:com.ibm.ejs.util.cache.CacheElementEnumerator=
all=enabled
```

   Set **Maximum File Size** to *200MB* or more. If you leave the default value of 20MB, you could fill up the single 20MB trace log and lose some data because of trace wrapping.

   Set **Maximum Number of Historical Files** to *5*. Five files should be sufficient, but if you see that all five files are full and trace wrapping occurs, increase this value.
2. Stop your server, delete existing logs, then start your server.
3. Run typical scenarios to capture cache trace data. By running a typical scenario with the trace enabled, you will get the EJB cache trace data to analyze in the following steps.
4. View and analyze the trace output.
   a. Open your trace log. Look for either or both of the following trace strings to appear:

      BackgroundLru 3 EJB Cache: Sweep (1,40) - Cache limit not reached : 489/2053
      BackgroundLru > EJB Cache: Sweep (16,40) - Cache limit exceeded : 3997/2053 Entry

      In the trace strings that include the words **Cache limit** you find a ratio. For example, 3997/2053. The first number is the number of enterprise beans currently in the EJB cache (called the *capacity*). The second number is the EJB cache setting (more about this in later steps). You will use this ratio, particularly the capacity, in your analysis.

      Also look for the statements *Cache limit not reached* and *Cache limit exceeded*.

      Cache limit not reached
      Your cache is equal to or larger than what is appropriate. If it is larger, you are wasting memory, and should reduce the cache size to a more appropriate value, as described below.

      Cache limit exceeded
      The number of beans currently being used is greater than the capacity you have specified,

indicating that your cache is not properly tuned. The capacity can exceed the EJB Cache setting because the setting is not a hard limit. The EJB Container does not stop adding beans to the cache when the limit is reached. Doing so could mean that when the cache is full, a request for a bean would not be fulfilled, or would at least be delayed until the cache fell below the limit. Instead, the cache limit can be exceeded, but the EJB Container attempts to clean up the cache and keep it below the EJB Cache size.

In the case where the cache limit is exceeded, you might see a trace point similar to this:

BackgroundLru < EJB Cache: Sweep (64,38) - Evicted = 50 : 3589/2053 Exit

Notice the *Evicted =* string. If you see this string, you are using either Stateful Session Beans or Entity Beans configured for Option A or B caching. Evicted objects mean you are not taking full advantage of the caching option that you have chosen. Your first step is to try increasing the EJB Cache size. If continued running of your application results in more evictions, it means that the application is accessing or creating more new beans between EJB Cache sweeps than the cache can hold, and *NOT* re-using existing beans. At this point, you might want to consider using Option C caching for your entity beans, or checking your application to see if it is not removing Stateful Session Beans after they are no longer needed.

Note: Entity beans configured with Option C caching are only in the cache while in a transaction, and are required to be held in the cache for the entire transaction. Therefore, they are never evicted during a cache sweep, but are removed from the cache when the transaction completes. In addition, if you are using only Stateless Session Beans or Entity Beans with Option C caching (or both), then you might want to increase your EJB Cache *cleanup interval* to a larger number. The cleanup interval can be set as described in "EJB cache settings" on page 180. Stateless Session Beans are **NOT** in the EJB Cache, and since Entity Beans using Option C caching are never evicted by the caching (LRU) strategy, there is really no need to sweep very often. When using only Stateless Session Beans or Option C caching, you should only see ″Evicted = 0″ in the trace example shown above.

b. Analyze your trace log. Look for the trace string *Cache limit exceeded*.
   - You might find more than one instance of this string. Examine them all to find the highest capacity value of beans in the EJB Cache. Re-set your EJB Cache size to about 110% of this number. Setting the EJB Cache size is explained in a later step.
   - You might find no instances of this string. This means that you have not exceeded the capacity of the EJB Cache (which is your end goal), but not seeing it during your initial analysis could also mean that your cache is too large and using unnecessary memory. In this case, you still need to tune your cache by reducing the cache size until your cache limit is exceeded, then increasing it to the optimum value. Setting the EJB Cache size is explained in a later step.

   Your ultimate goal is to set the cache limit to a value that does not waste resources, but also does not get exceeded. So, a good set-up gives you a trace with only the *Cache limit not reached* message, and a ratio where the capacity number can be near, but below, 100% of the EJB Cache setting.

   Note: It is recommended that you do not set your cache size to anything less than the default of 2053.

5. Modify the cache settings based on your analysis. See "EJB cache settings" on page 180 for information on how to do this.
6. Stop your server, delete all logs, and restart your server.
7. Repeat the above steps until you are satisfied with your settings.
8. Disable the EJB Cache trace. With the cache properly tuned, you can remove the trace, remove old logs, and restart your server.

What to do next

From your analysis, it is possible to set the EJB cache optimally from an EJB Container perspective, but perhaps not optimally from a WebSphere Application Server perspective. A larger cache size provides more hits and better EJB cache performance, but uses more memory. Memory used by the cache is not available to other areas of the product, potentially causing the overall performance to suffer. In a system with ample memory, this might not be an issue and properly tuning the EJB cache might increase overall performance. However, you should take into account this system performance versus EJB cache performance when configuring the cache.

Related tasks

Working with trace
Use trace to obtain detailed information about running the WebSphere Application Server components, including application servers, clients, and other processes in the environment.

Related reference

"EJB cache settings"
Use this page to configure and manage the cache for a specific Enterprise JavaBeans (EJB) container. To avoid errors from attempting to overload the cache, determine the cache absolute limit. Multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. This value is the limit that the cache will hold. You can use the Tivoli Performance Viewer to view bean performance information.

EJB cache settings

Use this page to configure and manage the cache for a specific Enterprise JavaBeans (EJB) container. To avoid errors from attempting to overload the cache, determine the cache absolute limit. Multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then, add the number of active session bean instances. This value is the limit that the cache will hold. You can use the Tivoli Performance Viewer to view bean performance information.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → **server** → **EJB Container Settings** → **EJB cache settings**.

Cleanup interval

Specifies the interval at which the container attempts to remove unused items from the cache in order to reduce the total number of items to the value of the cache size. This setting applies to the cache only.

The cache manager tries to maintain some unallocated entries that can be allocated quickly as needed. A background thread attempts to free some entries while maintaining some unallocated entries. If the thread runs while the application server is idle, when the application server needs to allocate new cache entries, it does not pay the performance cost of removing entries from the cache. In general, increase this parameter as the cache size increases.Timeouts are specified according to the transaction type:

- Container-managed transactions: The bean provider configures the timeout attribute in the deployment desecriptor.
- Bean-managed transaction: An application calls the UserTransaction.setTransactionTimeout method in the codes.

Data type	Integer
Units	Milliseconds
Range	0 to 2 147 483 647
Default	3000

Cache size

Specifies the number of buckets in the active instance list within the EJB container.

A bucket can contain more than one active enterprise bean instance, but performance is maximized if each bucket in the table has a minimum number of instances assigned to it. When the number of active instances within the container exceeds the number of buckets, that is, the cache size, the container periodically attempts to reduce the number of active instances in the table by passivating some of the active instances. For the best balance of performance and memory, set this value to the maximum number of active instances expected during a typical workload.

Data type	Integer
Units	Buckets in the hash table
Range	Greater than 0. The container selects the next largest prime number equal to or greater than the specified value.
Default	2053

Container interoperability

Container interoperability describes the ability of the product clients and servers at different versions to successfully negotiate differences in native Enterprise JavaBeans (EJB) finder methods support and Java EE compliance.

Interoperability of the handle formats in WebSphere Application Server, Version 5 and Version 5.0.1

Applications that attempt to persist handles to enterprise beans and **EJBHome** needed to subclass ObjectInputStream in WebSphere Application Server, Version 5. This action was required so that the subclass ObjectInputStream could utilize the context class loader to resolve the classes for enterprise beans and EJBHome stubs.

In addition, handles created and persisted in WebSphere Application Server, Version 5 only work with objects that have an unchanged remote interface. If the remote interface is changed, the handle is no longer valid because the stub is serialized inside the handle and its serial Version UID changes if the remote interface changes.

This release introduces a new handle persistence mechanism that avoids the implementation drawbacks of the previous version. However, if handles are used for this WebSphere Application Server deployment, you should consider the following issues when applying this update, future WebSphere Application Server Fix Packs and EJB Container cumulative fixes for WebSphere Application Server, Version 5.

If a WebSphere Application Server, Version 5 persisted handle or home handle is encountered by a WebSphere Application Server, Version 5.0.1 system, it can be read and utilized. In addition, it will be converted to WebSphere Application Server, Version 5.0.1 format if it is re-persisted. The WebSphere Application Server, Version 5.0.1 format cannot be read by a WebSphere Application Server, Version 5 system unless PQ72184 is applied.

Problems arise when handles are persisted and shared across systems that are not at the WebSphere Application Server, Version 5.0.1 level or later. However, a Version 5 system can receive a handle from Version 5.0.1 remotely through a call to get a handle on an enterprise bean or a getHomeHandle on an **EJBHome**. The remote call will succeed, however, any attempt to persist it on the Version 5 system will have the same limitations regarding the use of ObjectInputStream and changes in remote interface invalidating the persisted handle.

When your application stores handles persistently and shares this persistence with multiple clients or application servers, apply WebSphere Application Server, Version 5.0.1 or PQ72184 to both the client and server systems at the same time. Failure to do so can result in the inability of these systems to read the handle data stored by upgraded systems. Also, handles stored by the WebSphere Application Server, Version 5 can force the applications of the updated system to still subclass `ObjectInputStream`.

Applications using the WebSphere Application Server Enterprise, Version 5 scheduler and process choreographer, are affected by these changes. These users should update their Version 5 systems at the same time with either Version 5.0.1 or PQ72184.

If the applications store handles in the session context, or locally in a file on the same system, that is not shared by other applications, on different systems, they might be able to update their systems individually, rather than all at once. If Client Container and thin client applications do not share persisted handle data, they can be updated as needed as well. However, handles created and persisted in WebSphere Application Server, Version 5, Version 4.0.3 and later (with the property flag set), or Version 3.5.7 and later (with the property flag set) are not usable if either the home or the remote interface changes.

If any WebSphere Application Server, Version 3.5.7 or Version 4.0.3 and later enables the system property com.ibm.websphere.container.portable to **true**, any handles to objects on that server have the same interoperability limitations. In addition, if any WebSphere Application Server, Version 3.5.7 and later or Version 4.0.3 applications store a handle obtained from a WebSphere Application Server, Version 5 or Version 5.0.1, the same restrictions apply, regarding the need to subclass ObjectInputStream and the usability of handles after a change to the remote interface is made.

Replication of the Http Session and Handles

This note applies to you if you place Handles to Homes or Enterprise JavaBeans, or EJB or EJBHome references in the Http Session in your application and you use Http Session Replication. If you intend to replicate a mixed environment of Version 5.0.0 and Version 5.0.1 or 5.0.2 machines you should first apply the latest Version 5.0.0 container cumulative e-fix to the Version 5.0.0 machines before allowing the Version 5.0.1 or 5.0.2 server into the typology. The reason for this is that Version 5.0.0 servers are not able to understand the persisted Handle format used on the Version 5.0.1 and 5.0.2 server. This is similar to the case of Version 5.0.0 and Version 5.0.1 or 5.0.2 systems trying to use a shared database, mentioned above. But in this case, it is the Http Session object and not the database providing the persistence.

Top Down Deployment Mapping

The size of the Handle objects has grown due to the fix put in to allow serialization and deserialization to occur without the previous requirements of subclassing the ObjectInputStream and so on. Top down deployment of an object that contains EJB and EJBHome references create a database table ddl that has a field of 1000 bytes of VARCHAR for BITDATA which will contain the Handle. It might be that your object's Handle does not fit in the 1000 byte default field, and you might need to adjust this to a higher value. You might try increments of 250 bytes, that is, 1250, 1500, and so on.

EJB 2.1 container tuning

If you use applications that affect the size of the Enterprise JavaBeans (EJB) container cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. Container managed persistence (CMP) is discussed in this topic, although it is important to know that entity beans are not supported in an EJB 3.0 module.

EJB cache size

Monitoring Tivoli Performance Viewer (TPV) is a great way to diagnose if the EJB container cache size setting is tuned correctly for your application.

If the application has filled the cache causing evictions to occur, TPV will show a very high rate of the ejbStores method being called and probably a lower than expected CPU utilization on the application server machine.

All applications using enterprise beans should have this setting adjusted from the default if the following formula works out to more than 2000.

```
EJB_Cache_Size = (Largest number of Option B or C Entity Beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A Entity Beans expected to be accessed during
typical application workload) +
(Number of stateful Session Beans active during typical workload) +
(Number of stateless SessionBean types used during typical workload)

Where:
Option B and C Entity Beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A Entity Beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there starts to become more beans in the cache than the cache
size has been set to. If your application uses Read Only Beans, consider them to be Option A beans for the purpose of this tuning calculation.

Stateful Session Beans are held in the EJB cache until they are removed by the
application, or their session timeout value is reached.

Only a single stateless Session Bean instance for each EJB type is held in the
cache during the time any methods are being executed on that stateless Session
Bean. If two or more methods are being executed simultaneously on the same
stateless Session Bean type, each method executes on its own bean instance, but
only one cache location is used for all of these instances.
```

This calculates the upper bound on the maximum possible number of enterprise beans active at one time inside the application server. Because the EJB container's cache is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the calculation above.

```
<tuning parameter>
This setting can be found under Servers > Application Servers > serverName >
         EJB Container > EJB Cache Settings
```

Also while adjusting the EJB cache size, the EJB container management thread parameter can be tuned to meet the needs of the application. The management thread is controlled through the Clean Up Interval setting. This setting controls how frequently a daemon thread inside of the product attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This ensures that the EJB container places and looks up items in the cache as quickly. It is typically best to leave this interval set to the default, however, in some cases, it might be worthwhile to see if there is a benefit to reducing this interval.

For information about tuning the EJB cache to an even finer degree using the EJB cache trace service, see "Tuning the EJB cache using the trace service" on page 178.

EJB stateful session bean tuning

Stateful session bean timeout values can be configured using an assembly tool. For WebSphere Application Server Version 5.1 and later, the default value is 600 seconds. If this default value is not acceptable, you can change it in the tooling's EJB deployment descriptor editor. Stateless session beans **do not** have a timeout value because they have no conversational state and are not dedicated to any specific client.

You **cannot** set the timeout value globally, or on the server, so that the value applies to all stateful session beans. You can only set the timeout on a bean-by-bean basis.

Dcom.ibm.websphere.ejbcontainer.poolSize

If the application is using the majority of the instances in the pool, the TPV indicates this. When this occurs, then the size of those bean pools that are being exhausted should be increased. This can be done by adding this parameter in the JVM's custom properties tag .

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>

where:
 <application_name> is the J2EE application name as defined in the application
archive (.ear) file deployment descriptor, for the bean whose pool size is being
set
```

```
<module_name> is the .jar file name of the EJB module, for the bean whose pool size
 is being set,

<bean_name> is the J2EE Enterprise Bean name as defined in the EJB module
deployment descriptor, for the bean whose pool size is being set

<minSize> is the number of bean instances the container maintains in the pool,
 irrespective of how long the beans have been in the pool (beans greater than this
 number are cleared from the pool over time to optimize memory usage)

<maxSize> is the number of bean instances in the pool where no more bean instances
 are placed in the pool after they are used (that is, once the pool is at this
 size, any additional beans are discarded rather than added into the pool --
this ensures the number of beans in the pool has an upper limit so memory usage
does not grow in an unbounded fashion).

To keep the number of instances in the pool at a fixed size, minSize and maxSize
can be set to the same number. Note that there is a separate instance pool for
every EJB type running in the application server, and that every pool starts out
with no instances in it - that is, the number of instances grows as beans are
used and then placed in the pool. When a bean instance is needed by the container
and no beans are available in the pool, the container creates a new bean
instance, uses it, then places that instance in the pool (unless there are
already maxSize instances in the pool).

For example, the statement
-Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327

would set a minSize of 125 and a maxSize of 1327 on the bean named "ivtEJBObject"
within the ivtEJB.jar file, in the application "ivtApp".
```

Where ivtApp is replaced by the actual application name, ivtEJB.jar file is replaced by the JAR that contains the bean that needs to have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size should be increased. The minimum number of beans that are held in the pool is 125. The maximum number of beans that are held in the pool is 1327. These should be set so no more evictions occur from the pool and in most cases should be set equal if memory is plentiful because no growth and shrinkage of the pool occurs.

Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation

You should have a good idea of how your application handles the creation of primary key objects for use by CMP beans and bean-managed persistence (BMP) beans inside of the product. The IBM EJB container uses the primary key of an entity bean as an identifier inside of many internal data structures to optimize performance. However, the EJB container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application, in case the application changes or mutates the primary key, to keep the internal structures consistent. If the application does not mutate any of the primary keys used to create and access entity beans after they are created, a special flag can be used that ensures the EJB container skips the copy of the primary key object, saving CPU cycles and increasing performance. This mechanism can be enabled at your own risk by adding the **–D** property to the JVM custom property field.

```
<tuning parameter>
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

The performance benefit of this optimization depends on the application. If the application uses primitive types for enterprise beans' primary keys there is no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys, that is, an object for a primary key or multiple fields, this parameter can yield significant improvements.

Dcom.ibm.ws.pm.deferredcreate

The persistence manager is used by the EJB container to persist data to the database from CMP entity beans. When creating entity beans by calling the ejbCreate method, by default the persistence manager

immediately inserts the empty row with only the primary key in the database. In most cases, after creating the bean you should modify fields in the bean created or in other beans inside of the same transaction. If you want to postpone the insert into the database until the end of the transaction to eliminate one trip to the database, set the **–D** flag inside of the JVM custom properties field. The data is inserted into the database and consistency is maintained.

```
<tuning parameter>
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are insert intensive the application can benefit from this optimization. If the application performs few inserts, the benefit of this optimization is less.

Dcom.ibm.ws.pm.batch

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans, such as updates, inserts, reads, the number of operations issued to the database corresponds directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements, you can enable this flag and increase performance on all interactions with the database that involve more than two updates in a single transaction.

This flag lets the persistence manager add all the update statements into one single batch statement that is issued to the database. This saves round trips to the database, increasing performance. If the you know that your application exhibits the behavior of updating multiple CMP beans in a single transaction, and the database supports batch updates, you can set the **–D** flag inside of the JVM custom properties field.

```
<tuning parameter>
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans or only updates a single bean per transaction there is no performance gain. If the application updates multiple beans per transaction, this parameter benefits the applications performance.

The following table lists the backend databases that support batch update.

Table 1.

Database	Supports batch update	Supports batch update with Optimistic Concurrency Control
DB2	yes	no
Oracle	yes	no
DB2 Universal Driver	yes	yes
Informix	yes	yes
SQLServer	yes	yes
Apache Derby	yes	yes

Note: Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

com.ibm.ws.pm.useLegacyCache

Specifies the name of the Java class that the product uses to implement the javax.rmi.CORBA.UtilDelegate interface.

Persistence manager has two types of caching mechanisms, *legacy cache* and *two-level cache*. Typically, two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

`com.ibm.ws.pm.useLegacyCache=false`

com.ibm.ws.pm.grouppartialupdate and com.ibm.ws.pm.batch

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence manager has two caching mechanisms available, legacy cache and two-level cache. Typically, two-level cache performs better than legacy cache because of the optimizations in this mode.

In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both:

`'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'`

Deploying EJB modules

When you deploy an Enterprise JavaBeans (EJB) module, you install that module on a server that has been configured to support deployed modules.

Before you begin

Assemble one or more EJB modules, assemble one or more Web modules, and assemble them into a J2EE application.

Assemble one or more EJB 3.0 modules, assemble one or more Web modules, and assemble them into a J2EE application.

For an overview about the changes to the EJB deployment model for EJB 3.0, see the topic ″EJB 3.0 deployment overview.″

1. Prepare the deployment environment.
2. Update the configuration for each EJB module as needed for the deployment environment.
3. Deploy the application.

What to do next

If you specify that the EJBDeploy tool be run during application installation and the installation fails with a NameNotFoundException message, ensure that the input Java archive (JAR) or enterprise archive (EAR) file does not contain source files. Either remove the source files or include all dependent classes and resource files on the class path. If there are source files in the input JAR or EAR file, the EJB deployment tools runs a rebuild before generating the deployment code.

If the module deploys successfully, test and debug the module.

EJB 3.0 deployment overview

Learn about the Enterprise JavaBeans (EJB) 3.0 deployment model, including to *Just-In-Time* (JIT) deployment.

All Java Platform, Enterprise Edition (Java EE) application server products have some form of EJB deployment phase where your application is customized to run in that particular application server implementation. Typically, this is accomplished by an application server-specific deployment tool, which generates code to bridge your EJB interface and implementation code to the application server's EJB

container implementation. Some application server products' deploy tools alter the bytecodes of your application classes rather than using code generation, but the end result is similar.

In WebSphere Application Server, the bridging is accomplished by generating code that *wraps* your EJB implementation classes, connecting them to the product EJB container, which in turn allows the EJB container to host your enterprise beans and provide services to them. If one or more of your enterprise beans has defined remote interfaces, additional code is generated to provide the remote function.

Historically, EJB deployment in the WebSphere product has been performed by the *EJBDeploy* tool that is included with product and packaged with WebSphere product-oriented development tools.

The EJBDeploy tool introspects your EJB external interfaces, generates the wrapper code as .java files, then compiles it using the javac compiler to produce .class files, which are then packaged in your EJB module with your application code. The EJBDeploy tool also runs the rmic tool against the remote EJB interfaces in the application, producing additional *stub* and *tie* class files that interact with the Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) Object Request Broker (ORB), providing remote object support. Typically, you run the EJBDeploy tool either when you install the application on the product or sometime before you install the application from the command-line tool or within a development tool.

Just-In-Time deployment

The EJB 3.0 support in WebSphere Application Server adds a new feature called Just-In-Time Deployment. With Just-In-Time Deployment, the EJB container dynamically generates the wrapper, stub, and tie classes in-memory as needed when the application is running. Additionally, the Web container and application client containers dynamically generate the stub class required for remote EJB invocations. Effectively, this means that you do not need to process EJB 3.0 modules, Web modules that invoke EJB 3.0 beans, or client modules that invoke EJB 3.0 beans, through the EJBDeploy tool prior to running them in WebSphere.

createEJBStubs tool

Even though the Just-In-Time Deployment feature, in many cases, dynamically generates the RMI-IIOP stub classes that are required for invocation of remote EJB interfaces, there remain some cases where these stub classes are not dynamically generated. For EJB 3.0 clients not running inside a Web container, EJB container, or client container, that is upgraded to EJB 3.0 level, you must generate the stub classes with the createEJBStubs tool, then make the generated stubs available in the client environment's classpath. Typically you would accomplish this by copying the generated stubs to the location where the client's business interface class resides.

To summarize, the createEJBStubs tool must be used to generate client-side stubs for the following environments:

- ″Bare″ Java Standard Edition (SE) clients, where a Java SE Java Virtual Machine (JVM) is the client environment.
- WebSphere Application Server container environments prior to Version 7 that do not have the Feature Pack for EJB 3.0 applied.
- Non-WebSphere Application Server environments.

For more information about packaging your EJB module, see the topic, ″EJB 3.0 module packaging overview.″

EJBDEPLOY relationships – troubleshooting tips

Use this information to troubleshoot information for EJBDEPLOY problems.

The converter that is defined for the primary key is not invoked on its foreign key value

The mapping for primary key fields to database columns may use a converter to transform the key values. If a container-managed persistence (CMP) bean uses a converter to map its primary key, and that bean has a relationship where the bean at the other end holds a foreign key, the mapping for the foreign key will not use the converter.

The following errors might occur, indicating that the converter defined for the primary key is not invoked on its foreign key value. During the run of the ejbDeploy command , you receive the following message:

```
No type mapping defined for Java datatype1 to Database datatype2
```

During run time, the application does not find the CMP bean at the other end of the relationship.

To work around this limitation, define your own foreign key in the database table, and create a mapping that uses the same converter as defined for the primary key on the enterprise beans at the other end of its relationship.

EJB module settings

Use this page to configure and manage a specific deployed EJB module.

Note: You cannot start or stop an individual EJB module for modification. You must start or stop the appropriate application entirely.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → **application_name** → **Manage Modules** → **module_name**.

Note: If an application is running, changing an application setting causes the application to restart. On stand-alone servers, the application restarts after you save the change. On multiple-server products, the application restarts after you save the change and files synchronize on the node where the application is installed. To control when synchronization occurs on multiple-server products, deselect **Synchronize changes with nodes** on the Console preferences page.

URI

Specifies location of the module relative to the root of the application EAR file. The URI must match the URI of a ModuleRef URI in the deployment descriptor of the deployed application (EAR).

Alternate deployment descriptor

Specifies an alternate deployment descriptor for the module as defined in the application deployment descriptor according to the J2EE specification.

Starting weight

Specifies the order in which modules are started when the server starts. The module with the lowest starting weight is started first.

Data type	Integer
Default	5000
Range	Greater than 0

Chapter 4. Client applications

Using application clients

An application client module is a Java Archive (JAR) file that contains a client for accessing a Java application.

About this task

Complete the following steps for developing different types of application clients.

1. Decide on a type of application client.
2. Develop the application client code.
   a. Develop ActiveX application client code.
   b. Develop J2EE application client code.
   c. Develop pluggable application client code.
   d. Develop thin application client code.
3. Assemble the application client using the Application Server Toolkit.
4. Deploy the application client.
   Deploy the application client on Windows systems.
5. Run the application client.

Example

The IBM Application Client for WebSphere Application Server Samples gallery is not available on WebSphere Application Server for i5/OS. To access these samples, install WebSphere Application Server Clients on the Windows platform, and retrieve the samples from your local file system as the following command indicates:

`<app_server_root>/samples/index.html`

Application Client for WebSphere Application Server

In a traditional client-server environment, the client requests a service and the server fulfills the request. Multiple clients use a single server. Clients can also access several different servers. This model persists for Java clients except that now these requests use a client runtime environment.

In this model, the client application requires a servlet to communicate with the enterprise bean, and the servlet must reside on the same machine as the WebSphere Application Server.

The Application Client for WebSphere Application Server Version 7.0 (Application Client) consists of the following client applications:

- Java Platform, Enterprise Edition (Java EE) application client application (Uses services provided by the Java EE Client Container)
- Thin application client application (Does not use services provided by the Java EE Client Container)

The Application Client is packaged with the following components:

- The application client on WebSphere Application Server for i5/OS provides the necessary Java extensions to work with the iSeries server Java (TM) Development Kit. The i5/OS product does not include a standalone JRE.
- WebSphere Application Server run time for Java EE application client applications or Thin application client applications

The *Java EE application client* is a Java application program that accesses enterprise beans, Java DataBase Connectivity (JDBC) APIs, and Java Message Service message queues. The Java EE application client program runs on client machines. This program follows the same Java programming model as other Java programs; however, the Java EE application client depends on the Application Client run time to configure its execution environment, and uses the Java Naming and Directory Interface (JNDI) name space to access resources.

The Thin application client provides a lightweight Java client programming model. This client is useful in situations where the client application requires a thinner, more lightweight environment than the one offered by the Java EE application client.

The Java EE application client programming model provides the benefits of the Java EE platform for the Java client application. Use the Java EE application client to develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere Application Server platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the Java EE platform, you can put the client application code from one Java EE platform implementation to another. The client application package can require redeployment using each Java EE platform deployment tool, but the code that comprises the client application remains the same.

The Application Client run time supplies a container that provides access to system services for the client application code. The client application code must contain a main method. The Application Client run time invokes this main method after the environment initializes and runs until the Java virtual machine code terminates.

The Java EE platform supports the Application Client use of *nicknames* or *short names*, defined within the client application deployment descriptor. These deployment descriptors identify enterprise beans or local resources (JDBC, Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the client application code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the Application Client can require redeployment.

The Application Client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The Application Client run time also provides support for security authentication to enterprise beans and local resources.

The Application Client uses the Java Remote Method Invocation-Internet InterORB Protocol (RMI-IIOP). Using this protocol enables the client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the Java EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the Java EE and CORBA environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

Application client functions

This topic provides information about available functions in the different types of clients.

Use the following table to identify the available functions in the different types of clients.

Available functions	ActiveX client	Applet client	J2EE client	Pluggable client	Thin client

Provides all the benefits of a J2EE platform	Yes	No	Yes	No	No
Portable across all J2EE platforms	No	No	Yes	No	No
Provides the necessary run-time support for communication between a client and a server	Yes	Yes	Yes	Yes	Yes
Supports the use of nicknames in the deployment descriptor files. **Note:** Although you can edit deployment descriptor files, do not use the administrative console to modify them.	Yes	No	Yes	No	No
Supports use of the RMI-IIOP protocol	Yes	Yes	Yes	Yes	Yes
Browser-based application	No	Yes	No	No	No
Enables development of client applications that can access enterprise bean references and CORBA object references	Yes	Yes	Yes	Yes	Yes
Enables the initialization of the client application run-time environment	Yes	No	Yes	No	No
Supports security authentication to enterprise beans	Yes	Limited	Yes	Yes	Yes
Supports security authentication to local resources	Yes	No	Yes	No	No
Requires distribution of application to client machines	Yes	No	Yes	Yes	Yes
Enables access to enterprise beans and other Java classes through Visual Basic, VBScript, and Active Server Pages (ASP) code	Yes	No	No	No	No
Provides a lightweight client suitable for download	No	Yes	No	Yes	Yes
Enables access JNDI APIs for enterprise bean resolution	Yes	Yes	Yes	Yes	Yes
Runs on client machines that use the Sun Java Runtime Environment	No	No	No	Yes	No
Supports CORBA services (using CORBA services can render the application client code nonportable)	No	No	Yes	No	No
Supports JMS connections to the default messaging provider	No	No	Yes	No	Yes
Supports JMS connections to the default messaging provider	No	No	Yes	Yes	Yes

ActiveX application clients

WebSphere Application Server provides an ActiveX to EJB bridge that enables ActiveX programs to access enterprise beans through a set of ActiveX automation objects.

The bridge accomplishes this access by loading the Java virtual machine (JVM) into any ActiveX automation container such as Visual Basic, VBScript, and Active Server Pages (ASP).

There are two main environments in which the ActiveX to EJB bridge runs:

- **Client applications**, such as Visual Basic and VBScript, are programs that a user starts from the command line, desktop icon, or Start menu shortcut.
- **Client services**, such as Active Server Pages, are programs started by some automated means like the Services control panel applet.

The ActiveX to EJB bridge uses the Java Native Interface (JNI) architecture to programmatically access the JVM code. Therefore the JVM code exists in the same process space as the ActiveX application (Visual Basic, VBScript, or ASP) and remains attached to the process until that process terminates. To create JVM code, an ActiveX client program calls the XJBInit() method of the XJB.JClassFactory object. For more information about creating JVM code for an ActiveX program, see ActiveX to EJB bridge, initializing JVM code.

After an ActiveX client program has initialized the JVM code, the program calls several methods to create a proxy object for the Java class. When accessing a Java class or object, the real Java object exists in the JVM code; the automation container contains the proxy for that Java object. The ActiveX program can use the proxy object to access the Java class, object fields, and methods. For more information about using Java proxy objects, see ActiveX to EJB bridge, using Java proxy objects. For more information about calling methods and access fields, see ActiveX to EJB bridge, calling Java methods and ActiveX to EJB bridge, accessing Java fields.

The client program performs primitive data type conversion through the COM IDispatch interface (use of the IUnknown interface is not directly supported). Primitive data types are automatically converted between native automation types and Java types. All other types are handled automatically by the proxy objects For more information about data type conversion, see ActiveX to EJB bridge, converting data types.

Any exceptions thrown in Java code are encapsulated and thrown again as a COM error, from which the ActiveX program can determine the actual Java exceptions. For more information about handling exceptions, see ActiveX to EJB bridge, handling errors.

The ActiveX to EJB bridge supports both free-threaded and apartment-threaded access and implements the free threaded marshaler (FTM) to work in a hybrid environment such as Active Server Pages. For more information about the support for threading, see ActiveX to EJB bridge, using threading.

Applet clients

The applet client provides a browser-based Java run time capable of interacting with enterprise beans directly, instead of indirectly through a servlet.

This client is designed to support users who want a browser-based Java client application programming environment that provides a richer and more robust environment than the one offered by the **Applet > Servlet > enterprise bean** model.

The programming model for this client is a hybrid of the Java application thin client and a servlet client. When accessing enterprise beans from this client, the applet can consider the enterprise bean object references as CORBA object references.

No tooling support exists for this client to develop, assemble or deploy the applet. You are responsible for developing the applet, generating the necessary client bindings for the enterprise beans and CORBA objects, and bundling these pieces together to install or download to the client machine. The Java applet

client provides the necessary run time to support communication between the client and the server. The applet client run time is provided through the Java applet browser plug-in that you install on the client machine.

Generate client-side bindings using an assembly tool. An applet can utilize these bindings, or you can generate client-side bindings using the **rmic** command. This command is part of the IBM Developer Kit, Java edition that is installed with the WebSphere Application Server.

The applet client uses the RMI-IIOP protocol. Using this protocol enables the applet to access enterprise bean references and CORBA object references, but the applet is restricted in using some supported CORBA services.

If you combine the enterprise bean and CORBA environments in one applet, you must understand the differences between the two programming models, and you must use and manage each model appropriately.

The applet environment restricts access to external resources from the browser run-time environment. You can make some of these resources available to the applet by setting the correct security policy settings in the WebSphere Application Server `client.policy` file. If given the correct set of permissions, the applet client must explicitly create the connection to the resource using the appropriate API. This client does not perform initialization of any service that the client applet can need. For example, the client application is responsible for the initialization of the naming service, either through the CosNaming, or the Java Naming and Directory Interface (JNDI) APIs.

Java EE application clients

The Java Platform, Enterprise Edition (Java EE) application client programming model provides the benefits of the Java EE Platform for WebSphere Application Server Enterprise product.

The Java EE platform offers the ability to seamlessly develop, assemble, deploy and launch a client application. The tooling provided with the WebSphere platform supports the seamless integration of these stages to help the developer create a client application from start to finish.

When you develop a client application using and adhering to the Java EE platform, you can put the client application code from one Java EE platform implementation to another. The client application package can require redeployment using each Java EE platform deployment tool, but the code that comprises the client application does not change.

The Java EE application client run time supplies a container that provides access to system services for the application client code. The Java EE application client code must contain a main method. The Java EE application client run time invokes this main method after the environment initializes and runs until the Java virtual machine application terminates.

Application clients can use *nicknames* or *short names*, defined within the client application deployment descriptor with the Java EE platform. These deployment descriptors identify enterprise beans or local resources (Java Database Connectivity (JDBC) data sources, J2C connection factories, Java Message Service (JMS), JavaMail and URL APIs) for simplified resolution through JNDI use. This simplified resolution to the enterprise bean reference and local resource reference also eliminates changes to the application client code, when the underlying object or resource either changes or moves to a different server. When these changes occur, the application client can require redeployment. Although you can edit deployment descriptor files, do not use the administrative console to modify them.

Note: Connection pool is not supported in application client. The application client calls the database directly, without a datasource. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational® Application Developer (RAD) or assembly tool. The connection is established between application

client and the database. Application client does not have connection pool, but JDBC provider configuration is available in the client deployment descriptors.

The Java EE application client also provides initialization of the run-time environment for the client application. The deployment descriptor defines this unique initialization for each client application. The Java EE application client run time also provides support for security authentication to the enterprise beans and local resources.

The Java EE application client uses the Java Remote Method Invocation technology run over Internet Inter-Orb Protocol (RMI-IIOP). Using this protocol enables the client application to access enterprise bean references and to use Common Object Request Broker Architecture (CORBA) services provided by the Java EE platform implementation. Use of the RMI-IIOP protocol and the accessibility of CORBA services assist users in developing a client application that requires access to both enterprise bean references and CORBA object references.

When you combine the Java EE and the CORBA WebSphere Application Server Enterprise environments or programming models in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

Pluggable application clients

The Pluggable application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

Note: The Pluggable application client is available only on the Windows platform.

The Pluggable application client requires that you have previously installed the Sun Java Runtime Environment (JRE) files. In all other aspects, the Pluggable application client, and the Thin application client are similar.

This client is designed to support those users who want a lightweight Java client application programming environment, without the overhead of the Java Platform, Enterprise Edition (Java EE) platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model, but supports access to enterprise beans.

When accessing enterprise beans from this client, the client application can consider the enterprise beans object references as CORBA object references.

Tooling does not exist on the client; however, tooling does exists on the server. You are responsible for developing the client application, generating the necessary client bindings for the enterprise bean and CORBA objects, and after bundling these pieces together, installing them on the client machine.

The Pluggable application client provides the necessary run time to support the communication needs between the client and the server.

The Pluggable application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access enterprise bean references and CORBA object references and use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can assist a user in developing a client application that needs to access both enterprise bean references and CORBA object references.

When you combine the Java EE and CORBA environments in one client application, you must understand the differences between the two programming models to use and manage each appropriately.

The Pluggable application client run time provides the necessary support for the client application for object resolution, security, Reliability Availability and Serviceability (RAS), and other services. However, this client does not support a container that provides easy access to these services. For example, no

support exists for using *nicknames* for enterprise beans or local resource resolution. When resolving to an enterprise bean (using either the Java Naming and Directory Interface (JNDI) API or CosNaming) sources, the client application must know the location of the name server and the fully qualified name used when the reference was bound into the name space.

When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, Java Message Service (JMS), and so on). This client does not perform initialization of any of the services that the client application might require. For example, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI APIs.

The Pluggable application client offers access to most of the available client services in the Java EE application client. However, you cannot access the services in the Pluggable application client as easily as you can in the Java EE application client. The Java EE client has the advantage of performing a simple JNDI name space lookup to access the desired service or resource. The Pluggable application client must code explicitly for each resource in the client application. For example, looking up an enterprise bean Home object requires the following code in a Java EE application client:

```
java.lang.Object ejbHome =
 initialContext.lookup("java:/comp/env/ejb/MyEJBHome");
MyEJBHome =
 (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);
```

However, you need more explicit code in a Pluggable application client for Java:

```
java.lang.Object ejbHome =
 initialContext.lookup("the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");
MyEJBHome =
 (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome, MyEJBHome.class);
```

In this example, the Java EE application client accesses a logical name from the `java:/comp` name space. The Java EE client run time resolves that name to the physical location and returns the reference to the client application. The pluggable client must know the fully qualified physical location of the enterprise bean Home object in the name space. If this location changes, the pluggable client application must also change the value placed on the lookup() statement.

In the Java EE client, the client application is protected from these changes because it uses the logical name. A change can require a redeployment of the EAR file, but the actual client application code remains the same.

The Pluggable application client is a traditional Java application that contains a *main* function. The WebSphere Pluggable application client provides run-time support for accessing remote enterprise beans, and provides the implementation for various services. This client can also access CORBA objects and CORBA-based services. When using both environments in one client application, you need to understand the differences between the enterprise bean and the CORBA programming models to manage both environments.

For instance, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model. Use the JNDI implementation in the enterprise bean model to initialize the Object Request Broker (ORB). The client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

The Pluggable application client provides a batch command that you can use to set the CLASSPATH and JAVA_HOME environment variables to enable the Pluggable application client run time.

Thin application clients

The Thin application client provides a lightweight, downloadable Java application run time capable of interacting with enterprise beans.

WebSphere Application Server supports the pluggable client.

The Thin application client is designed to support those users who want a lightweight Java client application programming environment, without the overhead of the Java Platform, Enterprise Edition (Java EE) platform on the client machine. The programming model for this client is heavily influenced by the CORBA programming model, but supports access to enterprise beans.

When accessing enterprise beans from this client, the client application can consider the enterprise beans object references as CORBA object references.

Tooling does not exist on the client, it exists on the server. You are responsible for developing the client application, generating the necessary client bindings for the enterprise bean and CORBA objects, and bundling these pieces together to install on the client machine.

The Thin application client provides the necessary runtime to support the communication needs between the client and the server.

The Thin application client uses the RMI-IIOP protocol. Using this protocol enables the client application to access not only enterprise bean references and CORBA object references, but also allows the client application to use any supported CORBA services. Using the RMI-IIOP protocol along with the accessibility of CORBA services can assist a user in developing a client application that needs to access both enterprise bean references and CORBA object references.

When you combine the Java EE and CORBA environments in one client application, you must understand the differences between the two programming models, to use and manage each appropriately.

The Thin application client run time provides the necessary support for the client application for object resolution, security, Reliability Availability and Servicability (RAS), and other services. However, this client does not support a container that provides easy access to these services. For example, no support exists for using *nicknames* for enterprise beans or local resource resolution. When resolving to an enterprise bean (using either Java Naming and Directory Interface (JNDI) or CosNaming) sources, the client application must know the location of the name server and the fully qualified name used when the reference was bound into the name space. When resolving to a local resource, the client application cannot resolve to the resource through a JNDI lookup. Instead the client application must explicitly create the connection to the resource using the appropriate API (JDBC, Java Message Service (JMS), and so on). This client does not perform initialization of any of the services that the client application might require. For example, the client application is responsible for the initialization of the naming service, either through CosNaming or JNDI APIs.

The Thin application client offers access to most of the available client services in the Java EE application client. However, you cannot access the services in the thin client as easily as you can in the Java EE application client. The Java EE client has the advantage of performing a simple Java Naming and Directory Interface (JNDI) name space lookup to access the desired service or resource. The thin client must code explicitly for each resource in the client application. For example, looking up an enterprise bean Home requires the following code in a Java EE application client:

```
java.lang.Object ejbHome =  initialContext.lookup("java:comp/env/ejb/MyEJBHome");
MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

However, you need more explicit code in a Thin application client:

```
java.lang.Object ejbHome =
  initialContext.lookup("the/fully/qualified/path/to/actual/home/in/namespace/MyEJBHome");
MyEJBHome = (MyEJBHome)javax.rmi.PortableRemoteObject.narrow(ejbHome,  MyEJBHome.class);
```

In this example, the Java EE application client accesses a logical name from the `java:comp` name space. The Java EE client run time resolves that name to the physical location and returns the reference to the client application. The Thin application client must know the fully qualified physical location of the enterprise bean Home in the name space. If this location changes, the thin client application must also change the value placed on the lookup() statement.

In the Java EE client, the client application is protected from these changes because it uses the logical name. A change might require a redeployment of the EAR file, but the actual client application code remains the same.

The Thin application client is a traditional Java application that contains a *main* function. The WebSphere Thin application client provides run-time support for accessing remote enterprise beans, and provides the implementation for various services. This client can also access CORBA objects and CORBA based services. When using both environments in one client application, you need to understand the differences between the enterprise bean and CORBA programming models to manage both environments.

For instance, the CORBA programming model requires the CORBA CosNaming name service for object resolution in a name space. The enterprise beans programming model requires the JNDI name service. The client application must initialize and properly manage these two naming services.

Another difference applies to the enterprise bean model. Use the Java Naming and Directory Interface (JNDI) implementation in the enterprise bean model to initialize the Object Request Broker (ORB). The client application is unaware that an ORB is present. The CORBA model, however, requires the client application to explicitly initialize the ORB through the `ORB.init()` static method.

The Thin application client provides a batch command that you can use to set the CLASSPATH and JAVA_HOME environment variables to enable the Thin application client run time.

WebSphere Application Server Version 6.1 and the Application client for WebSphere Application Server Version 6.1 also provides specialized types of Thin application client. The Web Services Thin Client is an unmanaged, stand-alone Java client environment that allows you to run JAX-RPC Web services client applications to invoke Web services that are hosted by the application server. The Administration Thin Client enables client applications to perform administration tasks outside of the application server environment.

Application client troubleshooting tips

This topic provides debugging tips for resolving common Java 2 Platform Enterprise Edition (J2EE) application client problems. To use this troubleshooting guide, review the trace entries for one of the J2EE application client exceptions, and then locate the exception in the guide.

Some of the errors in the guide are samples, and the actual error you receive can be different than what is shown here. You might find it useful to rerun the `launchClient` command specifying the `-CCverbose=true` option. This option provides additional information when the J2EE application client run time is initializing.

Error: java.lang.NoClassDefFoundError

Explanation This exception is thrown when Java code cannot load the specified class.

Possible causes
- Invalid or non-existent class
- Class path problem
- Manifest problem

Recommended response

Check to determine if the specified class exists in a Java Archive (JAR) file within your Enterprise Archive (EAR) file. If it does, make sure the path for the class is correct. For example, if you get the exception:

```
java.lang.NoClassDefFoundError:
WebSphereSamples.HelloEJB.HelloHome
```

verify that the HelloHome class exists in one of the JAR files in your EAR file. If it exists, verify that the path for the class is WebSphereSamples.HelloEJB.

If both the class and path are correct, then it is a class path issue. Most likely, you do not have the failing class JAR file specified in the client JAR file manifest. To verify this situation, perform the following steps:

1. Open your EAR file with an assembly tool, and select the Application Client.
2. Add the names of the other JAR files in the EAR file to the Classpath field.

This exception is generally caused by a missing Enterprise Java Beans (EJB) module name from the Classpath field.

If you have multiple JAR files to enter in the Classpath field, be sure to separate the JAR names with spaces.

If you still have the problem, you have a situation where a class is loaded from the file system instead of the EAR file. This error is difficult to debug because the offending class is not the one specified in the exception. Instead, another class is loaded from the file system before the one specified in the exception. To correct this error, review the class paths specified with the -CCclasspath option and the class paths configured with the Application Client Resource Configuration Tool.

Look for classes that also exist in the EAR file. You must resolve the situation where one of the classes is found on the file system instead of in the `.ear` file. Remove entries from the classpaths, or include the `.jar` files and classes in the `.ear` file instead of referencing them from the file system.

If you use the -CCclasspath parameter or resource classpaths in the tool, and you have configured multiple JAR files or classes, verify they are separated with the correct character for your operating system. Unlike the Classpath field, these class path fields use platform-specific separator characters.
Note: The system class path is not used by the Application Client run time if you use the launchClient batch or shell files. In this case, the system class path would not cause this problem. However, if you load the launchClient class directly, you do have to search through the system class path as well.

Error: com.ibm.websphere.naming.CannotInstantiateObjectException: Exception occurred while attempting to get an instance of the object for the specified reference object. [Root exception is javax.naming.NameNotFoundException: xxxxxxxxxx]

Explanation

This exception occurs when you perform a lookup on an object that is not installed on the host server. Your program can look up the name in the local client Java Naming and Directory Interface (JNDI) name space, but received a NameNotFoundException exception because it is not located on the host server. One typical example is looking up an EJB component that is not installed on the host server that you access. This exception might also occur if the JNDI name you configured in your Application Client module does not match the actual JNDI name of the resource on the host server.

Possible causes

- Incorrect host server invoked
- Resource is not defined
- Resource is not installed
- Application server is not started
- Invalid JNDI configuration

Recommended response

If you are accessing the wrong host server, run the `launchClient` command again with the -CCBootstrapHost parameter specifying the correct host server name. If you are accessing the correct host server, use the product `dumpnamespace` command line tool to see a listing of the host server JNDI name space. If you do not see the failing object name, the resource is either not installed on the host server or the appropriate application server is not started. If you determine the resource is already installed and started, your JNDI name in your client application does not match the global JNDI name on the host server. Use the Application Server Toolkit to compare the JNDI bindings value of the failing object name in the client application to the JNDI bindings value of the object in the host server application. The values must match.

Error: javax.naming.ServiceUnavailableException: A communication failure occurred while attempting to obtain an initial context using the provider url: ″iiop://[invalidhostname]″. Make sure that the host and port information is correct and that the server identified by the provider URL is a running name server. If no port number is specified, the default port number 2809 is used. Other possible causes include the network environment or workstation network configuration. Root exception is org.omg.CORBA.INTERNAL: JORB0050E: In Profile.getIPAddress(), InetAddress.getByName[invalidhostname] threw an UnknownHostException. minor code: 4942F5B6 completed: Maybe

Explanation

This exception occurs when you specify an invalid host server name.

Possible causes

- Incorrect host server invoked
- Invalid host server name

Recommended response

Run the `launchClient` command again and specify the correct name of your host server with the -CCBootstrapHost parameter.

Error: javax.naming.CommunicationException: Could not obtain an initial context due to a communication failure. Since no provider URL was specified, either the bootrap host and port of an existing ORB was used, or a new ORB instance was created and initialized with the default bootstrap host of ″localhost″ and the default bootstrap port of 2809. Make sure the ORB bootstrap host and port resolve to a running name server. Root exception is org.omg.CORBA.COMM_FAILURE: WRITE_ERROR_SEND_1 minor code: 49421050 completed: No

Explanation

This exception occurs when you run the `launchClient` command to a host server that does not have the Application Server started. You also receive this exception when you specify an invalid host server name. This situation might occur if you do not specify a host server name when you run the launchClient tool. The default behavior is for the launchClient tool to run to the local host, because WebSphere Application Server does not know the name of your host server. This default behavior only works when you are running the client on the same machine with WebSphere Application Server is installed.

Possible causes

- Incorrect host server invoked
- Invalid host server name
- Invalid reference to `localhost`
- Application server is not started
- Invalid bootstrap port

Recommended response

If you are not running to the correct host server, run the `launchClient` command again and specify the name of your host server with the `-CCBootstrapHost` parameter. Otherwise, start the Application Server on the host server and run the `launchClient` command again.

Error: javax.naming.NameNotFoundException: Name comp/env/ejb not found in context ″java:″

Explanation

This exception is thrown when the Java code cannot locate the specified name in the local JNDI name space.

Possible causes

- No binding information for the specified name
- Binding information for the specified name is incorrect
- Wrong class loader was used to load one of the program classes
- A resource reference does not include any client configuration information

Recommended response

Open the EAR file with the Application Server Toolkit, and check the bindings for the failing name. Ensure this information is correct. If you are using Resource References, open the EAR file with the Application Client Resource Configuration Tool, and verify that the Resource Reference has client configuration information and the name of the Resource Reference exactly matches the JNDI name of the client configuration. If the values are correct, you might have a class loader error. For detailed information about the configuration tool and opening EAR files, read the Starting the Application Client Resource Configuration Tool in the *Developing and deploying applications* PDF book.

Error: java.lang.ClassCastException: Unable to load class: org.omg.stub.WebSphereSamples.HelloEJB._HelloHome_Stub at com.ibm.rmi.javax.rmi.PortableRemoteObject.narrow(portableRemoteObject.java:269)

Explanation

This exception occurs when the application program attempts to narrow to the EJB home class and the class loaders cannot find the EJB client side bindings.

Possible causes

- The files, *_Stub.class and _Tie.class, are not in the EJB `.jar` file
- Class loader could not find the classes

Recommended response

Look at the EJB `.jar` file located in the `.ear` file and verify the class contains the Enterprise Java Beans (EJB) client side bindings. These are class files with file names that end in `_Stub` and `_Tie`. If the binding classes are in the EJB `.jar` file, then you might have a class loader error.

Error: WSCL0210E: The Enterprise archive file [EAR file name] could not be found. com.ibm.websphere.client.applicationclient.ClientContainerException: com.ibm.etools.archive.exception.OpenFailureException

Explanation

This error occurs when the application client run time cannot read the Enterprise Archive (EAR) file.

Possible causes

The most likely cause of this error is that the system cannot find the EAR file in the path specified on the `launchClient` command.

Recommended response

Verify that the path and file name specified on the `launchclient` command are correct.

The launchClient command appears to hang and does not return to the command line when the client application has finished.

Explanation

When running your application client using the `launchClient` command the WebSphere Application Server run time might need to display the security login dialog. To display this dialog, WebSphere Application Server run time creates an Abstract Window Toolkit (AWT) thread. When your application returns from its main method to the application client run time, the application client run time attempts to return to the operating system and end the Java virtual machine (JVM) code. However, since there is an AWT thread, the JVM code will not end until `System.exit` is called.

Possible causes

The JVM code does not end because there is an AWT thread. Java code requires that `System.exit()` be called to end AWT threads.

Recommended response

- Modify your application to call `System.exit(0)` as the last statement.
- Use the `-CCexitVM=true` parameter when you call the `launchClient` command.

IBM Support has documents and tools that can save you time gathering information needed to resolve problems as described in Troubleshooting help from IBM. Before opening a problem report, see the Support page:

- http://www.ibm.com/servers/eserver/support/iseries/allproducts/index.html

clientUpgrade script

The clientUpgrade script migrates application client modules and their resources in an enterprise archive (EAR) file so that these application clients can run in WebSphere Application Server Version 7. The script converts an EAR file that you want to migrate and then overwrites the original EAR file with the converted EAR file.

The following title provides information about the clientUpgrade script.

Note: This command was deprecated in Version 6.1.

Type	Description
Product	The clientUpgrade script is available in the WebSphere Application Server (Express and Base) product only.
Authority	To run this script, your user profile must have *ALLOBJ authority.

Syntax	The syntax of the clientUpgrade script is:
	`clientUpgrade` *`EAR_file`* `[ -clientJAR` *`client_JAR_file`* `]`
	`[ -logFileLocation` *`logFileLocation`* `]`
	`[ -traceString` *`trace_spec`* `[ -traceFile` *`file_name`* `] ]`

Parameters	The parameters of the clientUpgrade script are: • *EAR_file* -- This is a required parameter. The value EAR_file specifies the fully-qualified path of the EAR file that contains the application client modules that you want to migrate. • -clientJAR -- This is an optional parameter. The value client_JAR_file specifies a JAR file that you want to migrate. The script overwrites the original EAR file with a new EAR file that contains only the specified JAR files. If you do not specify this parameter, the clientUpgrade script migrates all client JAR files in the EAR file. • -logFileLocation -- Use this optional parameter to specify an alternate location to store the log output. • -traceString -- This is an optional parameter. The value trace_spec specifies the trace information that you want to collect. To gather all trace information, specify ″*=all=enabled″ (including the double quotation marks (″)). By default, the script does not gather trace information. If you specify this parameter, you must also specify the -traceFile parameter. • -traceFile -- This is an optional parameter. The value file_name The value file_name specifies the name of the output file for trace information. If you specify the -traceString parameter but do not specify the -traceFile parameter, the script does not generate a trace file.
Logging	The clientUpgrade script displays status while it runs. It also saves more extensive logging information to the clientupgrade.log file. This file is located in the /QIBM/UserData/WebSphere/AppServer/V7/edition/profiles/default/logs directory (for a default installation using the default profile) or in the location specified by the -logFileLocation parameter.

These examples demonstrate correct syntax. In this example, the My51Application.ear file is migrated from WebSphere Application Server Version 5.1, The script overwrites the original EAR file with a new file that you can deploy in your WebSphere Application Server Version 7 profile.

```
clientUpgrade /My51Application/My51Application.ear
```

In this example, only the `myJarFile.jar` client JAR file is migrated. The script overwrites `My51Application.ear` with an EAR file that contains `myJarFile.jar`. You can deploy the new EAR file in your WebSphere Application Server profile.

```
clientUpgrade /My51Application/My51Application.ear -clientJAR myJarFile.jar
```

Running application clients

The Java Platform, Enterprise Edition (Java EE) specification requires support for a client container that runs stand-alone Java applications (known as Java EE application clients) and provides Java EE services to the applications. Java EE services include naming, security, and resource connections.

About this task

You are ready to run your application client using this tool after you:
1. Write the application client program.
2. Assemble and install an application module (`.ear` file) in the application server run time.
3. Deploy the application using the Application Client Resource Configuration Tool (ACRCT) on Windows.

This task only applies to Java EE application clients. To launch Java EE application clients using the launchClient script, perform the following steps:

1. Start the Qshell environment.

   On the CL command line, enter the command:

   ```
   STRQSH
   ```

2. Enter the following command to launch Java EE application clients:

   app_server_root`/bin/launchClient`

   where *app_server_root* is /QIBM/ProdData/WebSphere/AppServer/V7/Base or /QIBM/ProdData/WebSphere/AppServer/V7/ND

3. Pass parameters to the `launchClient` command or to your application client program as well. The `launchClient` command allows you to do both. The `launchClient` command requires that the first parameter is either:
   - An EAR file specifying the application client to launch.
   - A request for `launchClient` usage information.

   The following example illustrates the command line invocation syntax for the launchClient tool:

   ```
   launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
   ```

   where

   - *userapp* is the path and the name of the EAR file that contains the application client.
   - *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
   - *app args* are arguments that pass to the application client.
   - *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.
   - *-JVMOptions* is a valid Java standard or non-standard option string. Insert quotation marks around the string.
   - *-help, -?* prints the usage information.

   All other parameters intended for the `launchClient` command must begin with the -CC prefix.

   Parameters that are not EAR files, or usage requests, or that do not begin with the -CC prefix, are ignored by the application client run time, and are passed directly to the application client program.

   The `launchClient` command retrieves parameters from three places:
   - The command line
   - A properties file
   - System properties

   The parameters are resolved in the order listed above, with command line values having the highest priority and system properties the lowest. Using this prioritization you can set and override default values.

4. Specify the server name.

   By default, the **launchClient** command uses *your_server_name* for the `BootstrapHost` property value.

   This setting is effective for testing your application client when it is installed on the same computer as the server. However, in other cases override this value with the name of your server. You can override the `BootstrapHost` value by invoking `launchClient` command with the following parameters:

   ```
   launchClient myapp.ear -CCBootstrapHost=abc.midwest.mycompany.com
   ```

   You can also override the default by specifying the value in a properties file and passing the file name to the launchClient shell.

   Security is controlled by the server. You do not need to configure security on the client because the client assumes that security is enabled. If server security is not enabled, then the server ignores the security request, and the application client functions as expected.

Example

You can store launchClient values in a properties file, which is a good method for distributing default values. You can then override one or more values on the command line. The format of the file is one `launchClient -CC` parameter per line without the `-CC` prefix. For example:

```
 verbose=true classpath=/usr/lpp/mydir/util.jar;/usr/lpp/mydir/harness.jar;/usr/lpp
/production/G19/global.jar BootstrapHost=abc.westcoast.mycompany.com tracefile=/usr
/lpp/WebSphere/mylog.txt
```

launchClient tool

This topic describes the Java Platform, Enterprise Edition (Java EE) command line syntax for the launchClient tool for WebSphere Application Server.

Note: All users who run commands from a specific profile must have authority to modify files that are created by other users that use the same profile. Otherwise, you might see a permission denied error in the log files. To avoid this issue, consider one of the following policies:

- Use specific profiles for distinct user authorities
- Always use the same user for all of the commands that are run in a given profile
- Ensure that all users of a specific profile belong to the same group. In addition, ensure that each user of a group has the read and write authority to the files that are created by other members in the same profile.

The following example illustrates the command line invocation syntax for the launchClient tool:

```
launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
```

where

- *userapp* is the path and the name of the EAR file that contains the application client.
- *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
- *app args* are arguments that pass to the application client.
- *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.
- *-JVMOptions* is a valid Java standard or nonstandard option string, except `-cp` or `-classpath`. Insert quotation marks around the string.
- *-help, -?* prints the usage information.

The first parameter must be `-help`, `-?` or contain no parameter at all. The `-profileName pName` and `-JVMOptions` options are optional parameters. If used, they must appear before the *<userapp>* parameter.

All other parameters are optional and can appear in any order after the *<userapp>* parameter. The Java EE Application client run time ignores any optional parameters that do not begin with a `-CC` prefix and passes those parameters to the application client.

Client container parameters

Supported arguments include:

-CCadminConnectorHost
Specifies the host name of the server from which configuration information is retrieved.

The default is the value of the `-CCBootstrapHost` parameter or the value, `your.server.name`, if the `-CCBootstrapHost` parameter is not specified.

-CCadminConnectorPort
Indicates the port number for the administrative client function to use. The default value is `8880` for SOAP connections and `2809` for Remote Method Invocation (RMI) connections.

-CCadminConnectorType
Specifies how the administrative client connects to the server. Specify `RMI` to use the RMI connection type, or specify `SOAP` to use the SOAP connection type. The default value is `SOAP`.

-CCadminConnectorUser
Administrative clients use this user name when a server requires authentication. If the connection type is SOAP, and security is enabled on the server, this parameter is required.

-CCadminConnectorPassword
The password for the user name that the `-CCadminConnectorUser` parameter specifies.

-CCaltDD
The name of an alternate deployment descriptor file. This parameter is used with the `-CCjar` parameter to specify the deployment descriptor to use. Use this argument when a client JAR file is configured with more than one deployment descriptor. Set the value to `null` to use the client JAR file standard deployment descriptor.

-CCBootstrapHost
The name of the host server you want to connect to initially. The format is:
your_server_of_choice.com

-CCBootstrapPort
The server port number. If you do not specify this argument, the WebSphere Application Server default value is `used`.

-CCclassLoaderMode
Specifies the class loader mode. If PARENT_LAST is specified, the class loader loads classes from the local class path before delegating the class loading to its parent. The classes loaded for the following are affected:

- Classes defined for the Java EE application client
- Resources defined in the Java EE application
- Classes specified on the manifest of the Java EE client JAR file
- Classes specified using the -CCclasspath option

If PARENT_LAST is not specified, then the default mode, PARENT_FIRST, causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path.

-CCclasspath
A class path value. When you launch an application, the system class path is used. If you want to access classes that are not in the EAR file or part of the system class paths, specify the appropriate class path here. Multiple paths can be concatenated.

-CCD
Use this option to have the WebSphere Application Server set the specified system property during initialization. Do not use the equals (=) character after the `-CCD`. For example: `-CCDcom.ibm.test.property=testvalue`. You can specify multiple `-CCD` parameters. The general format of this parameter is `-CCD<property key>=<property value>`. For example, `-CCDI18NService.enable=true`.

-CCdumpJavaNameSpace
Controls generation of a dump of the java: name space for the application that is launched, which can be used for debugging purposes. A value of **true** generates a dump in short format, and includes the name and object type for each binding. A value of **long** generates a dump in long format, and includes additional information for each binding over short format, such as the local object type and string representation of the local object. The default value is **false**, and does not generate a dump.

-CCexitVM
Use this option to have the WebSphere Application Server call the `System.exit()` method after the client application completes. The default is `false`.

-CCinitonly
Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is `false`.

-CCjar
The name of the client Java Archive (JAR) file that resides within the EAR file for the application you wish to launch. Use this argument when you have multiple client JAR files in the EAR file.

-CCpropfile
Indicates the name of a properties file that contains launchClient properties. Specify the properties without the `-CC` prefix in the file, with the exception of the securityManager, securityMgrClass and securityMgrPolicy properties. See the following example: `verbose=true`.

-CCproviderURL
Provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a Common Object Request Broker Architecture (CORBA) object URL or an Internet Inter-ORB Protocol (IIOP) URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. You can specify bootstrap server addresses, for all servers in the cluster, in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the `-CCBootstrapHost` and `-CCBootstrapPort` parameters. A CORBA object URL specifying multiple systems is illustrated in the following example:

```
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
```

This value is mapped to the `java.naming.provider.url` system property.

-CCsecurityManager
Enables and runs the WebSphere Application Server with a security manager. The default is `disable`.

-CCsecurityMgrClass
Indicates the fully qualified name of a class that implements a security manager. Only use this argument if the `-CCsecurityManager` parameter is set to `enable`. The default is `java.lang.SecurityManager`.

-CCsecurityMgrPolicy
Indicates the name of a security manager policy file. Only use this argument if the `-CCsecurityManager` parameter is set to enable. When you enable this parameter, the `java.security.policy` system property is set. The default is *<app_server_root>*/ `properties/client.policy`.

-CCsoapConnectorPort
The Simple Object Access Protocol (SOAP) connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCtrace
Use this option to obtain debug trace information. You might need this information when reporting a problem to IBM customer support. The default is `false`. For more information, read the topic ″Enabling trace.″

-CCtracefile
Indicates the name of the file to which trace information is written. The default is to write output to the console.

-CCtraceMode
Specifies the trace format to use for tracing. If the valid value, `basic`, is not specified the default is advanced. Basic tracing format is a more compact form of tracing.

For more information on basic and advanced trace formatting, see Interpreting trace output.

-CCverbose
This option displays additional information messages. The default is `false`.

If you are using an EJB client application with security enabled, edit the `sas.client.props` file, which is located in the *profile_root*`/properties` directory. Within the file, change the `com.ibm.CORBA.loginSource` value to *none*.

For more information on the `sas.client.props` utility, see Manually encoding passwords in properties files and the PropFilePasswordEncoder command reference.

RMI connection with security. Used with the EJB and administrative client application.

Using Jacl:

```
wsadmin -conntype RMI -port  rmiportnumber -user userid
-password password
```

Using Jython:

```
wsadmin -lang jython -conntype RMI -port  rmiportnumber -user userid
-password password
```

rmiportnumber for your connection displays in the administrative console as BOOTSTRAP_ADDRESS.

Note: On the AIX®, HP-UX, Linux®, i5/OS, Solaris, and z/OS operating systems, the use of -password option may result in security exposure as the password information becomes visible to the system status program, such as ps command, which can be invoked by other users to display all of the running processes. Do not use this option if security exposure is a concern. Instead, specify user and password information in the `soap.client.props` file for SOAP connector or `sas.client.props` file for RMI connector. The `soap.client.props` and `sas.client.props` files are located in the properties directory of your WebSphere Application Server profile.

If Kerberos (KRB5) is enabled for administrative authentication, the authentication target supports BasicAuth and KRB5. To use KRB5, update the `sas.client.props`, `soap.client.props`, and `ipc.client.props` files, according to the connector type.

Note: When using Kerberos authentication, the user password does not flow across the wire. A one-way hash of password is used to identify the client.

The following examples demonstrate correct syntax.

/QIBM/ProdData/WebSphere/AppServer/V61/Base/bin/launchClient /home/earfiles/myapp.ear -profileName myprofile -CCBootstrapHost=myWASServer -CCverbose=true app_parm1 app_parm2

Specifying the directory for an expanded EAR file

You can archive the `Manifest.mf` client Java Archive (JAR) files instead of automatically cleaning them up after the application exits.

Before you begin

Each time the launchClient tool is called, it extracts the Enterprise Archive (EAR) file to a random directory name in the temporary directory on your hard drive. Then the tool sets up the thread ClassLoader to use the extracted EAR file directory and JAR files included in the `Manifest.mf` client Java Archive (JAR) file. In a normal J2EE Java client, these files are automatically cleaned up after the application exits. This cleanup occurs when the client container shutdown hook is called. To avoid extracting the EAR file (and removing the temporary directory) each time the launchClient tool is called, complete the following steps:

1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
2. Delete the directory before running the launchClient tool again, if you need to update your EAR file. When you call the `launchClient` command, it extracts the new EAR file to the directory. If you do not delete the directory or change the system property value to point to a different directory, the launchClient tool reuses the currently extracted EAR file and does not use your changed EAR file. When specifying the `com.ibm.websphere.client.applicationclient.archivedir` property, make sure that the directory you specify is unique for each EAR file you use. For example, do not point the `MyEar1.ear` and the `MyEar2.ear` files to the same directory.

Using Java Web Start

Learn about the Java Web Start technology that is provided by the Java Standard Edition runtime environment to deploy Java Enterprise Edition application clients, including Thin application clients, on the remote client machine with a single click from Web browser on the client machine.

Before you begin

The supported client platforms for deploying application clients using the Java Web Start are the same as the IBM Application Client for WebSphere Application Server supported platforms, except Linux on Power and OS/400 operating systems.

Before you begin this task, see the following topics to understand Java Web Start technology and its components:

- “Java Web Start architecture for deploying application clients” on page 210
- “Client application Java Network Launcher Protocol deployment descriptor file” on page 211
- “ClientLauncher class” on page 215

You can use the following resources:

- The IBM implementation of Java Web Start on Java Platform Standard Edition 6 (Java SE 6) that is packaged in the Application Client for WebSphere Application Server.

Note: When using the Sun JRE with the Application Client runtime, application resources cannot be downloaded using the https protocol. This is a Sun Microsystems limitation. Install the IBM JRE on the client machine and use IBM Java Web Start to run the WebSphere Application Client application. Sun Java Web Start is supported, but with a limitation that only the http protocol can be used to download the application resource, including the application jnlp description file.

About this task

To deploy application clients using Java Web Start, the client machine must have at least a Java SE runtime environment installed. The Java SE runtime environment includes the Java Web Start, which implements the JSR 56: Java Network Launching Protocol and API. The application clients Enterprise Archive (EAR) file is a Java archive (JAR) resource in a JNLP descriptor file that resides on a central server. The JNLP descriptor file also specifies the runtime environment requirement for running the application.

WebSphere Application Server provides a launcher class to launch the Java EE application client in the application client container inside of Java Web Start. The client machine might not have the IBM Application Client for WebSphere Application Server installed. If this is the case, create and install an application client container and runtime package as a runtime environment through Java Web Start. The JNLP descriptor file specifies this runtime environment as the required runtime environment for running the Java EE application client.

WebSphere Application Server also provides command-line utility programs to create this application client container and runtime package from an existing IBM Application Client for WebSphere Application Server installation, as well as an installer class to install this package as a runtime environment for the application client container and also the Java Runtime Environment (JRE) in the IBM Application Client for WebSphere Application Server installation. To run the Java EE application client, the EAR file is deployed as a JAR resource that is described in the JNLP descriptor file.

1. Identify the client machine operating system, and install the corresponding IBM Application Client for WebSphere Application Server on a development machine. For example, if the Java EE application clients are targeted to run on Windows operating systems, install the IBM Application Client for WebSphere Application Server for Windows. Follow the instructions for “Installing Application Client for WebSphere Application Server” on page 242.
2. Run the utility programs to create the application client container and runtime package.
   a. Use the “buildClientRuntime tool” on page 218 utility to create the package.
   b. Use the “buildClientLibJars tool” on page 211 utility to create the JAR files containing the launcher and the installer class. This utility also zips up the properties files in the <app_client_root>/ properties directory.
3. Create the runtime installer JNLP descriptor file. The JNLP response must be included in the JNLP version ID to indicate the current runtime version in the response header, for example, `x-java-jnlp-version-id=1.6.0`. Using a servlet of a Java Server Pages (JSP) file to provide a dynamic JNLP response.
4. Create the Java EE application client launch JNLP descriptor file.
5. Package the application client container runtime environments and the Java EE application in an Enterprise Archive (EAR) file. Depending on your preferred deployment strategy, the files can be in two separate Web modules, or combined into one.
6. All JAR resources must be Java signed, including the Java EE application client EAR file.
7. Deploy the Enterprise Archive file on an application server, and start the application. The Java EE application client is ready to be deployed.

Example

A Java Web Start deployment Sample is included in the client samples. This Sample demonstrates the steps to deploy a Java EE application client with an automated ANT script. The Sample has a servlet to generate the runtime installer JNLP response with JNLP version ID, for example, `x-java-jnlp-version-id`.

Note: When the application client initially launches using Java Web Start from Sun Microsystems Java SE Runtime Environment 6.0, it installs the Application Client runtime, which includes the IBM JRE. An null pointer exception (NPE) is thrown from the `com.sun.deploy.services.WPlatformService.getSecureRandom()` method. This is a known bug in

Sun Java SE 6 (http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6505528). If you experience this exception, relaunch the application. The NPE only occurs on the first launch of the application client.

Java Web Start architecture for deploying application clients

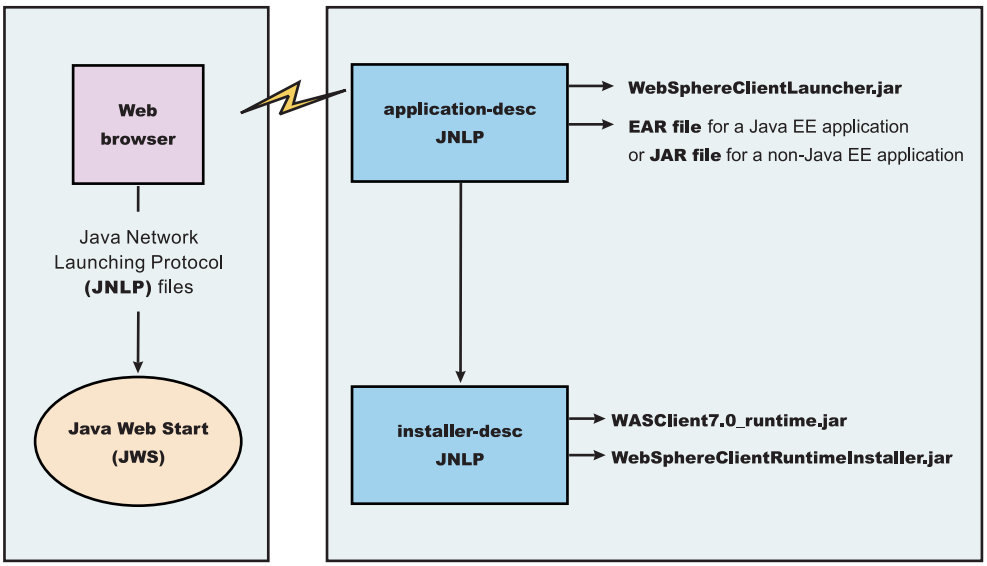
Java Web Start is an application-deployment technology that includes the portability of applets, the maintainability of servlets and JavaServer Pages (JSP) file technology, and the simplicity of mark-up languages such as XML and HTML. It is a Java application that allows full-featured Java EE client applications to be launched, deployed and updated from a standard Web server. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported.

Upon launching Java Web Start for the first time, you might download new client applications from the Web. Each time you launch JWS thereafter, you can initiate applications either through a link on a Web page or (in Windows) from desktop icons or the Start menu. You can deploy applications quickly using Java Web Start, cache applications on the client machine, and launch applications remotely offline. Additionally, because Java Web Start is built from the Java Platform, Enterprise Edition (Java EE) infrastructure, the technology inherits the complete security architecture of the Java EE platform.

The technology underlying Java Web Start is the Java Network Launching Protocol & API (JNLP). Java Web Start is a JNLP client and it reads and parses a JNLP descriptor file (JNLP file). Based on the JNLP descriptor, it downloads appropriate pieces of a client application and any of its dependencies. If any of the pieces of the application are already cached on the client machine, then those components are not downloaded again, unless they have been updated on the server machine. After you download and cache the client application, JWS launches it natively on the client machine.

The following diagram shows an overview of launching a client application, include the Application Client for WebSphere Application Server as a dependent resource, using Java Web Start.



The Web browser running on a client machine connects to a Web application located on a server machine. The client application JNLP descriptor file is downloaded and processed by Java Web Start on the client machine.

In this diagram, there are two JNLP descriptor files:

- Client application JNLP descriptor (application-desc in the diagram)
- Application Clients run-time installer JNLP descriptor (installer-desc in the diagram)

Each of these JNLP descriptor files, the client application (JAR or EAR) and the dependent resource JAR files are packaged as Web applications in an EAR file. This EAR file is deployed to an Application server. The client machine with JWS installed uses a Web browser to connect to the url of the client application JNLP descriptor file to download and run the client application.

Using Java Web Start from Java SE Runtime Environment 6.0 or later is highly recommended. All the platforms supported by the application client for WebSphere Application Server are supported with the exception Linux on Power and OS/400 platforms.

You can use the following:

- Java Web Start on the Java Standard Edition Developer Kits that IBM provides, packaged in Application Client for WebSphere Application Server
- Java Web Start on Sun Microsystems Java SE 6 Development Kit or Java SE Runtime Environment 6.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java Platform, Standard Edition, version 6, which you can download from the HP Web site

buildClientLibJars tool:

For a Java Platform, Enterprise Edition (Java EE) application client application and or Thin application client application to be launched using Java Web Start (JWS), the properties files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to create those property JAR files. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported on WebSphere Application Server for i5/OS.

The buildClientLibJars tool copies the JAR files from the Application Client for WebSphere Application Server installation and creates a `properties.jar` file, which contains the properties files from the Application Clients installation properties directory to a specified location. When this property is created, the tool uses the value of `keystore`, `storepass`, `alias` and `storetype` to sign all of the JAR files in the specified location.

```
Windows usage: buildClientLibJars.bat  [-help] [-verbose] destdir keystore storepass alias storetype

Unix usage:    buildClientLibJars.sh  [-help] [-verbose] destdir keystore storepass alias storetype
```

where:

- `-help` will display the message
- `-verbose` will turn on verbose message
- `destdir` will output the destination directory name
- `keystore` is the key store file
- `storepass` is the key store password
- `alias` key is the alias name
- `storetype` is the key store type

Client application Java Network Launcher Protocol deployment descriptor file

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Location

The client application has an Application Clients run-time dependency that provides the following:

- Java SE Runtime Environment from IBM

- Application Clients run-time properties
- SSL KeyStore and TrustStore file
- Application Clients run-time library JAR files (optional for Thin Application client applications)

If the Application Clients run-time dependency is not met, it is downloaded and installed in Java Web Start (JWS), as described by the Application Clients run-time installer JNLP descriptor file. For example:

```
<j2se version="1.6" href="http://your_server.com/jws/wasappclient/download.jnlp"/>
```

Usage notes

The client application must also include the `WebSphereClientLauncher.jar` file, which contains the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, that completes one of the following actions:

- If it is a Java Platform, Enterprise Edition (Java EE) Application client application (that is the resources for the application contain an EAR file with a client application), the EAR file must be specified as a JAR resource so that it can be downloaded to JWS and specified in the system property, `com.ibm.websphere.client.launcher.ear`. See “JNLP descriptor file for a Java EE Application client application” on page 213 for an example.
- If it is a Thin Application client application, the Thin Application client application JAR file must be specified as a JAR resource so that it can be downloaded to JWS and the name of the class containing main method entry point is specified in the system property, `com.ibm.websphere.launcher.main`. See “JNLP descriptor file for a Thin Application client application” on page 213 for an example.

The JNLP specification requires all the resource (JAR or EAR) files used in a JNLP file to be signed.

You can specify the –CC arguments defined in the launchClient tool for a J2EE Application client application in application arguments section of the JNLP descriptor files. However, only –CCD is supported for a Thin Application client application to define system properties and the JNLP <property> tag can also be used to define system properties. See the following example for details:

```
<property name="java.naming.provider.url" value="corbaloc:iiop:myserver.com:9089"/>
```

For a J2EE Application client application, specify the following application arguments as defined in the JNLP.

1. Specify your target server provider URL, as shown in the following example:

   ```
   <argument> >-CCDjava.naming.provider.url =corbaloc:iiop:myserver.mydomain.com:9080 </argument>
   ```

2. Specify the SSL Key File and SSL Trust File location. These files are expected to be available in the client machine. To use the ones in the Application Clients run-time dependency installed in JWS cache, specify these application arguments:

   ```
   <argument> -CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12 </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12 </argument>
   ```

3. Specify the initial naming context factor, as shown in the following example:

   ```
   <argument>-CCDjava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory </argument>
   ```

   For a Thin Application client application, you also need to specify the actual location of the `sas.client.props` and `ssl.client.props` files located in the Application Clients run-time dependency that is installed in the JWS cache.

   ```
   <argument>-CCDcom.ibm.CORBA.ConfigURL=file:${WAS_ROOT}/properties/sas.client.props </argument>
   <argument>-CCDcom.ibm.SSL.ConfigURL=file:${WAS_ROOT}/properties/ssl.client.props </argument>
   ```

   If any of the default settings in the `sas.client.props` and `ssl.client.props` file need modifying, use the –CCD to change the settings through the system properties, as shown in the following example:

   ```
   <argument>-CCDjavacom.ibm.CORBA.securityEnabled=false </argument>
   ```

   Note: The ${WAS_ROOT} token used in the JNLP file is replaced by the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, to the actual location of the Application Clients run-time dependency installation in the JWS cache. If you are using JSP to dynamically

create this JNLP description file, you must escape this token because it has a different meaning in JSP 2.0. See the following example for details:

```
<argument>-CCDcom.ibm.ssl.keyStore=\${WAS_ROOT}/etc/key.p12 </argument>
<argument>-CCDcom.ibm.ssl.trustStore=\${WAS_ROOT}/etc/trust.p12 </argument>
```

JNLP descriptor file for a Java EE Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Here is an example of the client application JNLP descriptor file for a Java EE Application client application:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->

<jnlp spec="1.0+" codebase="http://your_server:port_number/jws/wasappclient/apps/">
 <information>
  <title>Java EE Client Example</title>
  <vendor>IBM</vendor>
  <homepage href="null"/>
  <description>Java WebStart example: Launching Java EE Application Client</description>
  <description kind="short">Java EE Applicaiton Client</description>
  <description kind="tooltip">Java EE Application Client</description>
 </information>

 <security>
  <all-permissions/>
 </security>

 <resources>
  <j2se href="http://your_server:port_number/jws/wasappclient/JREDownload.xjnlp" version="1.6"/>
  <jar href="../lib/WebSphereClientLauncher.jar" download="eager" main="false"/>
  <jar href="../lib/properties.jar" download="eager" main="false"/>
  <jar href="SwingCalculator.ear" download="eager" main="false"/>

  <property name="com.ibm.websphere.client.launcher.ear" value="SwingCalculator.ear"/>
 </resources>

 <application-desc main-class="com.ibm.websphere.client.launcher.ClientLauncher">
  <argument>-CCproviderURL=corbaloc:iiop:tiu03.torolab.ibm.com:2809</argument>
 </application-desc>

</jnlp>
```

JNLP descriptor file for a Thin Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

Here is an example of the JNLP descriptor file for a Thin Application client application.

```
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->

<!-
=========================================================================
->
<!-- TODO: change "codebase" to the actual URL location of the jnlp file ->
=========================================================================
->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://your_server:port_number/jws/wasappclient/apps">
  <information>
  <title>Thin Base Calculator Client Samples</title>
  <vendor>IBM</vendor>
  <description>Thin Base Calculator Client Samples</description>
  <offline-allowed/>
 </information>

 <security>
  <all-permissions/>
 </security>
  <resources>
   <j2se version="1.6" href="http://your_server:port_number/jws/wasappclient/JREDownload.xjnlp"/>

   <jar href="/jws/wasappclient/lib/WebSphereClientLauncher.jar" main="true"/>
   <jar href="BasicCalculatorClientCommon.jar"/>
   <jar href="BasicCalculatorEJB.jar"/>
   <jar href="BasicCalculatorThinClient.jar"/>

   <property name="com.ibm.websphere.client.launcher.main"
      value="com.ibm.websphere.samples.technologysamples.basiccalcthinclient.BasicCalculatorClientThinMain"/>
   <property name="java.naming.factory.initial"
              value="com.ibm.websphere.naming.WsnInitialContextFactory" />
   <property name="java.naming.provider.url"
```

```
                value="corbaloc:iiop:tiu03:2809"/>

   </resources>

   <add</argument>
    <argument>1</argument>
    <argument>2</argument>
   </application-desc>
</jnlp>
```

ClientLauncher class:

The class, com.ibm.websphere.client.installer.ClientLauncher, contains a main() method that is called by Java Web Start (JWS) to launch the client application. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported.

This client is packaged in the `WebSphereClientLauncher.jar` file that is located in the Application Client for WebSphere Application Server installation under the <*app_client_root*>/ lib/webstart directory.

The launcher class requires that the following properties are defined. These properties are not defined in a separate properties file. Instead, the properties are defined as part of the Java Network Launching Protocol (JNLP) files.

com.ibm.websphere.client.launcher.main
If the client application is a Thin Application client, then this property should be specified. It specifies the class where the main entry point of the client application resides.

com.ibm.websphere.client.launcher.ear
If the client application is a Java Platform, Enterprise Edition (Java EE) Application client, then this property should be specified. It specifies the name of the EAR file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main. However, only one of the two properties should be specified.

Application client launcher for Java Web Start:

The application client launcher for Java Web Start is a Java class, com.ibm.websphere.client.installer.ClientLauncher, which has a main() method that Java Web Start calls to start the application client container and to invoke the application client's main() method. It provides similar functions as the **lauchClient** command line tool to start application clients from the command line.

The com.ibm.websphere.client.launcher.ClientLauncher class is packaged in the WebSphereClientLauncher.jar file under the <app_client_root>/lib/webstart directory.

The launcher tool requires that the following properties are defined.

com.ibm.websphere.client.launcher.main
If the client that is to be run is a thin client, then this property should be specified. It specifies the class where the main entry point of the application resides. It is the main class name for a Thin application client. If it is set, the launcher will not start the client container, it will rather invoke the main method for the application directly. However, if `com.ibm.websphere.client.launcher.ear` is also set, it will be ignored.

com.ibm.websphere.client.launcher.ear
If the client that is to run is the Java Platform, Enterprise Edition (Java EE) client, then this property should be specified. It specifies the name of the ear file to be executed. This property takes precedence over `com.ibm.websphere.client.launcher.main` although only one of the two properties should be specified.

These properties are not defined in a separate properties file. Instead, they are defined as part of the Java Network Launching Protocol files.

When `com.ibm.websphere.client.launcher.ear` is set, the application client launcher for JWS supports almost all of the –CC arguments as the **lauchClient** command line tool supports. However, if only `com.ibm.websphere.client.launcher.main` is set, the launcher will only support the –CCD argument. The following table shows the comparison of the supported –CC arguments for the **launchClient** command line tool and the application client launcher for JWS:

-CC argument	launchClient	Application client launcher for JWS
-CCverbose	Yes	Yes
-CCjar	Yes	Yes
-CCclasspath	Yes	N/A
-CCadminConnectorHost	Yes	Yes
-CCadminConnectorPort	Yes	Yes
-CCadminConnectorType	Yes	Yes
-CCadminConnectorUser	Yes	Yes
-CCaltDD	Yes	Yes
-CCBootstrapHost	Yes	Yes
-CCBootstrapPort	Yes	Yes
-CCproviderURL	Yes	Yes
-CCinitonly	Yes	N/A
-CCtrace	Yes	Yes
-CCtracefile	Yes	Yes
-CCsecurityManager	Yes	N/A
-CCsecurityMgrClass	Yes	N/A
-CCsecurityMgrPolicy	Yes	N/A
-CCD	Yes	Yes
-CCexitVM	Yes	Yes
-CCdumpJavaNameSpace	Yes	Yes
-CCsoapConnectorPort	Yes	Yes
-CCtraceMode	Yes	Yes
-CCclassLoaderMode	Yes	Yes

Macro expansion is supported for the –CCD argument by the application client launcher for JWS. The launcher will automatically substitute certain macro keys (enclosed with ${...}) with the calculated value at runtime. For example, if a macro key is used in the –CCD argument in the application client JNLP manifest file,

```
<argument>-CCDcom.ibm.ssl.keyStore= ${WAS_ROOT}/etc/key.p12</argument>
```

it will be expanded to the JWS cache installation root location and the argument will become:

```
-CCDcom.ibm.ssl.keyStore=/home/tiu/.java/deployment/cache/javaws/ext/E1134532441112/etc/key12.p12
```

The following table shows the 3 macro keys that are currently supported and will be substituted by the launcher:

Table 2.

Macro key	Value
${WAS_ROOT}	Installation root location within the JWS cache that is used by the application client container and runtime installer for JWS.
${JAVA_HOME}	Location of Java home. The return value of System.getProperty("java.home").
${USER_HOME}	Location of user home. The return value of System.getProperty("user.home").

Preparing the application client run time dependency component for Java Web Start

To launch a Java Platform, Enterprise Edition (Java EE) application client application, a Thin application client application, or both using Java Web Start (JWS), a Java Runtime Environment implementation Java archive (JAR) that IBM provides, the library JAR files and properties files bundled in Application Client for WebSphere Application Server must be installed in the JWS. Learn the steps to build the application client run time dependency component from an application client installation. It is packaged as a Web Archive Resource (WAR) file that can be installed in an Application Server.

Before you begin

Install the Application Client for WebSphere Application Server for the operating system to which the client application deploys. If there is a requirement to deploy the client application to multiple operating systems, the application client run time dependency component must be built separately for each operating system that client application supports.

1. Install the Application Client for WebSphere Application Server for the client application supported operating systems.
2. Change the directory to the installation bin directory.
3. Run the "buildClientRuntime tool" on page 218 to generate the application client run time JAR file, which contains the Java Standard Edition Runtime Environment, the run time library JAR files, properties files, and the SSL KeyStore and TrustStore files from the application client installation.
4. Run the buildClientLibJars tools to package up the properties files in the properties directory of the application client installation into a properties.jar file in the specified location. The buildClientLibJars tools will also copy the `WebSphereClientLauncher.jar` file and `WebSphereClientRuntimeInstaller.jar` file from the application client installation to the specified location. All jar files in the specified location will be signed by the provided certificate.

   For example, if you are using Version 7.0 and using the test certificate that is included in the application client installation:

   ```
   buildClientLibJars C:\Temp\webstart ..\etc\DummyClientKeyFilejar WebAS "websphere dummy client" JKS
   ```
5. Create a JavaServer Pages (JSP) file or use a servlet to generate the application client run time installer Java Network Launching Protocol (JNLP) descriptor to respond to Java Web Start request. See the Java Web Start Deployment sample in the application client installation.
6. Package the two signed JAR files, `WASClient7.0_windows.jar` and `WebSphereClientRuntimeInstaller.jar`, and the JSP file or servlet for generating the Application Client run time installer JNLP descriptor into a Web archive (WAR) file. This WAR file is packaged into an EAR file that can be deployed to an application server. See the Java Web Start Deployment sample in the application client installation.

Results

Your Web application is ready to serve the application client run time and the JRE environment.

buildClientRuntime tool:

For a Java Platform, Enterprise Edition (Java EE) application client application and or Thin application client application to be launched using Java Web Start (JWS), the library JAR files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to build those JAR files. The Java Web Start client is used with platforms that support a Web browser.

The Java Web Start client is not supported on WebSphere Application Server for OS/400.

The buildClientRuntime tool builds the required components from the WebSphere Application Server clients installation into the JAR file specified on the command. This JAR file contains:

- License files
- Java SE Runtime Environment 6 (JRE 6) that IBM provides
- Application Clients runtime properties and configuration
- SSL KeyStore and TrustStore files
- Runtime library JAR files

In the case of building an Application Clients runtime JAR file only for serving Thin Application client applications and not for Java EE Application client applications, the runtime library JAR files and the Application Clients runtime properties files are not included, except the configuration files, `sas.client.props`, `ssl.client.props` and `soap.client.props`, located in the `WAS_ROOT/properties` directory. The Java Web Start client is used with platforms that support a Web browser.

The command-line invocation syntax for the buildClientRuntime tool is shown in the following example:

```
Windows Usage:    buildClientRuntime.bat  [-help] [-verbose] outfile keystore storepass alias storetype

Unix Usage:         buildClientRuntime.sh    [-help] [-verbose] outfile keystore storepass alias storetype
```

where:

- -help will display the message
- -verbose will turn on verbose message
- outfile is the output file name
- keystore is the key store file
- storepass is the key store password
- alias is the key alias name
- storetype is the key store type

ClientRuntimeInstaller class:

This section provides information on the ClientRuntimeInstaller class.

This class, com.ibm.websphere.client.installer.ClientRuntimeInstaller, contains a main() method that Java Web Start (JWS) calls to install the Application Client for WebSphere Application Server run-time dependency component in JWS cache. It is packaged in `WebSphereClientRuntimeInstaller.jar` file located in the Application Client for WebSphere Application Server installation in the *<app_server_root>*/`JWS` directory.

Specify the WebSphereClientRuntimeInstaller.jar file and the Application Client run-time dependency component JAR file as JAR resources in the Application Client run-time installer Java Network Launcher Protocol (JNLP) descriptor file. See the following example for details:

```
<jar href="Launcher/WebSphereClientRuntimeInstall.jar" main="true"/>
<jar href="Launcher/WASClient6.1_windows.jarRuntimeInstall.jar" main="true"/>
```

The ClientRuntimeInstaller class main method requires the following properties to be set in the JNLP file:

com.ibm.websphere.client.jre.version
Specifies a Java Runtime Environment (JRE) version name that is to be used when referring to the Application Client run-time dependency component.

com.ibm.websphere.client.jre.launch.java
Specifies the relative location of the javaw.exe program in the Application Client run-time dependency component JAR file.

The previously mentioned properties, JRE version name and the location of the javaw.exe program are registered to the Java Web Start Application Manager, as shown in the following example:

```
<property name="com.ibm.websphere.client.jre.version" value="WASclient6.1"/>
<property name="com.ibm.websphere.client.jre.launch.java" value="java\jre\bin\javaw.exe"/>
```

Using the Java Web Start sample

The EAR file, WebSphereClientRuntime.ear, is provided in the JWS directory of the Client Application for WebSphere Application Server installation. This EAR file provides a sample Application Clients run-time installer JNLP descriptor file and a sample Application Clients run-time library component JNLP descriptor file. Follow the steps in this task to build the Application Clients run-time dependency component and the Application Clients run-time library component. Add these components to the WebSphereClientRuntime.ear file, and then install the EAR file in an Application Server to be used by the client application.

About this task

There is a new Java Web Start sample available in the client sample gallery for WebSphere Application Server V7.0. Refer to the client sample gallery in the Application Client for WebSphere Application Server product. The name of the new sample is ″Java Web Start Deployment Sample″.

Installing Java Web Start

Learn about the steps that are necessary to install Java Web Start (JWS) on the AIX platform. Java Web Start technology is provided by the Java SE runtime environment to deploy Java EE application clients (including Thin application clients) on the remote client machine with a single click from Web browser on the client machine.

Before you begin

Note: This topic applies only to the AIX operating system.

Before you begin this task, see the “Preparing the application client run time dependency component for Java Web Start” on page 217 topic to understand Java Web Start (JWS) technology and components.

Note:

You can use the following resources:

- The IBM implementation of Java Web Start on Java Platform Standard Edition 6 (Java SE 6) that is packaged in the Application Client for WebSphere Application Server

Note: You can use Java Web Start on Java Platform, Standard Edition Developer Kit 6 that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 7.0; Java Web Start on Sun Microsystems Java Standard Edition Software Development Kit 6 or Java SE Runtime

Environment 6.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems, or the Java Web Start on HP Software Development Kit (SDK) or RTE for Java 2 version 5.0, which you can download from the HP Web site.

About this task

Use the following steps to install JWS on the AIX platform.

1. Install IBM Application Client for WebSphere Application Server.
2. Change your directory to the client_install_root/java/jre/lib/javaws path.
3. Run the updateSetting.sh script from the path mentioned in the previous step.
4. Change your path to the JWS installed path. For example, enter: `client_install_root/java/jre/javaws`.
5. Run `./javaws` from the path mentioned in the previous step.

Using a static JNLP file with Java Web Start for Application clients

Do not use JSP to dynamically generate a JNLP file, otherwise the JNLP jsp page cannot be opened in some IE browsers.

About this task

To use a static JNLP file, you will need to add the following mime type mapping in the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
       <display-name>
       WAS Client runtime for Java Web Start</display-name>
       <welcome-file-list>
              <welcome-file>index.html</welcome-file>
              <welcome-file>index.htm</welcome-file>
              <welcome-file>index.jsp</welcome-file>
              <welcome-file>default.html</welcome-file>
              <welcome-file>default.htm</welcome-file>
              <welcome-file>default.jsp</welcome-file>
       </welcome-file-list>
       <mime-mapping>
       <extension>jnlp</extension>
       <mime-type>application/x-java-jnlp-file</mime-type>
       </mime-mapping>
</web-app>
```

Running the IBM Thin Client for Enterprise JavaBeans (EJB)

An EJB Client is a Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) Java Platform, Standard Edition (Java SE) application that accesses remote Enterprise Java Beans from a server through Java Naming and Directory Interface (JNDI) look up. IBM Thin Client for EJB offers a smaller footprint and is easy to deploy to a Java SE environment and an Eclipse Rich Client Platform (RCP) environment. You can bundle the IBM Thin Client for EJB library using the WebSphere Application Server installation or the Application Client for WebSphere Application Server installation with your application. The IBM Thin Client for EJB also extends the choice of Java SE runtime. It can be run in the Java Runtime Environment (JRE) that is packaged with the WebSphere Application Server product, the Sun Microsystems JRE that is downloaded from the Sun Microsystems Web site, or the JRE that is downloaded from the HP Web site.

Before you begin

The IBM ORB implementation library is required if the IBM Thin Client for EJB is running with a non-IBM product JRE on a non-IBM product platform. For example, running the IBM Thin Client for EJB with Sun

Microsystems JRE on Windows, Linux, or Solaris, and with the HP JRE on HPUX. The IBM-provided Solaris hybrid and HP hybrid JRE are not considered non-IBM product JRE environments. The IBM Thin Client for EJB can access version 2.x and version 3.0 EJB on the WebSphere Application Server using the JNDI lookup, but it cannot access version 3.0 EJB through resource injection. Resource injection is supported if the client application is a Java Platform, Enterprise Edition (Java EE) Application Client running within the Java Platform, Enterprise Edition (Java EE) Application Client Container.

Before you set up an EJB Thin Client environment, obtain the Java archive (JAR) file for the EJB Thin Client for WebSphere Application Server. To obtain the EJB Thin Client for WebSphere Application Server, install WebSphere Application Server or Application Client. The EJB Thin Client for WebSphere Application Server file, `com.ibm.ws.ejb.thinclient_7.0.0.jar`, is located in the *app_server_root*`\runtimes` directory.

Copy the Java archive (JAR) file for the IBM Thin Client for EJB with WebSphere Application Server product, `com.ibm.ws.ejb.thinclient_7.0.0.jar` , to other machines to create a lightweight client environment that enables communications with the products. Copies of the IBM Thin Client for EJB are subject to the same terms and conditions of the license agreement for Version 7.0 WebSphere Application Server where you obtained the Thin Client for EJB. Refer to the license agreements for correct usage and other limitations.

The IBM Thin Client for EJB with WebSphere Application Server runs on distributed operating systems with JDK support, including both Version 5 and Version 6. When using the IBM Thin Client for EJB as a standalone Java SE application with a non-IBM product JRE, you must override the default ORB implementation for the JRE through one of following methods:

- Include the com.ibm.ws.orb_7.0.0.jar file in the Java system classpath.
- Override the default ORB implementation in the JRE, using Java Endorsed Standards Override Mechanism.
- Set the java.endorsed.dirs path to a directory that contains the com.ibm.ws.orb_7.0.0.jar file.

When running the IBM Thin Client for EJB as an Eclipse RCP application, it is recommended to use method two, to override the default JRE ORB implementation.

Note: The Pluggable Application Client feature in the application client installer is deprecated. It is replaced by the IBM Thin Client for EJB.

About this task

Run the IBM Thin Client for EJB, by completing the following steps.

1. Invoke the client application. Run the following Java command:

   Add the following system properties to the Java command if you want authentication and SSL enabled:
2. Provide IIOP authentication configuration and Client SSL Configuration. Add the following system properties to the Java command:

   ```
   -Dcom.ibm.SSL.ConfigURL=file:///home/user1/ssl.client.props
   -Dcom.ibm.CORBA.ConfigURL=file:///home/user1/sas.client.props
   ```

   You can obtain the `ssl.client.props` file and `sas.client.props` file from the WebSphere Application Server installation and modify the file to suit your environment. You must, at a minimum, update the location of the key files in the `ssl.client.props` file to the match location of your target environment. For example,

   ```
   -Dcom.ibm.ssl.keyStore=/home/user1/etc/key.p12
   -Dcom.ibm.ssl.trustStore=/home/user1/etc/trust.p12
   ```

   Recommended SSL configuration settings when running the application with a non-IBM product JRE:

   ```
   com.ibm.ssl.protocol=SSL
   com.ibm.ssl.trustManager=SunX509
   com.ibm.ssl.keyManager=SunX509
   com.ibm.ssl.contextProvider=SunJSSE
   ```

```
com.ibm.ssl.keyStoreType=JKS
com.ibm.ssl.keyStoreProvider=SUN
com.ibm.ssl.keyStore=/home/user1/etc/key.jks

com.ibm.ssl.trustStoreType=JKS
com.ibm.ssl.trustStoreProvider=SUN
com.ibm.ssl.trustStore=/home/user1/etc/trust.jks
```

The key store file and trust store file must be created using the Java keytool utility before the application runs. The automatic key file generation is not supported with a non-IBM product JRE.

You must override the default ORB implementation of the non-IBM product JRE with the `com.ibm.ws.orb_7.0.0.jar` file, or add it to the classpath.

What to do next

Enable trace for the IBM Thin Client for EJB by adding the following to the Java command.

```
-Dcom.ibm.ejs.ras.lite.traceSpecification=*=all
```

Related tasks

Example 1: Configuring basic authentication and identity assertion
This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user bob. The following steps take you through the configuration of C, S1, and S2.

Using JMS to connect to a WebSphere Application Server default messaging provider messaging engine

Related reference

ssl.client.props client configuration file
Use the `ssl.client.props` file to configure Secure Sockets Layer (SSL) for clients. In previous releases of WebSphere Application Server, SSL properties were specified in the `sas.client.props` or `soap.client.props` files or as system properties. By consolidating the configurations, WebSphere Application Server enables you to manage security in a manner that is comparable to server-side configuration management. You can configure the `ssl.client.props` file with multiple SSL configurations.

Running application clients

The Java Platform, Enterprise Edition (Java EE) specification requires support for a client container that runs stand-alone Java applications (known as Java EE application clients) and provides Java EE services to the applications. Java EE services include naming, security, and resource connections.

About this task

You are ready to run your application client using this tool after you:
1. Write the application client program.
2. Assemble and install an application module (`.ear` file) in the application server run time.
3. Deploy the application using the Application Client Resource Configuration Tool (ACRCT) on Windows.

This task only applies to Java EE application clients. To launch Java EE application clients using the launchClient script, perform the following steps:

1. Start the Qshell environment.

   On the CL command line, enter the command:

   ```
   STRQSH
   ```

2. Enter the following command to launch Java EE application clients:

   app_server_root`/bin/launchClient`

   where *app_server_root* is /QIBM/ProdData/WebSphere/AppServer/V7/Base or /QIBM/ProdData/WebSphere/AppServer/V7/ND

3. Pass parameters to the `launchClient` command or to your application client program as well. The `launchClient` command allows you to do both. The `launchClient` command requires that the first parameter is either:
   - An EAR file specifying the application client to launch.
   - A request for `launchClient` usage information.

   The following example illustrates the command line invocation syntax for the launchClient tool:

   ```
   launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
   ```

   where
   - *userapp* is the path and the name of the EAR file that contains the application client.
   - *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
   - *app args* are arguments that pass to the application client.
   - *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.
   - *-JVMOptions* is a valid Java standard or non-standard option string. Insert quotation marks around the string.
   - *-help, -?* prints the usage information.

   All other parameters intended for the `launchClient` command must begin with the -CC prefix.

   Parameters that are not EAR files, or usage requests, or that do not begin with the -CC prefix, are ignored by the application client run time, and are passed directly to the application client program.

   The `launchClient` command retrieves parameters from three places:
   - The command line
   - A properties file
   - System properties

   The parameters are resolved in the order listed above, with command line values having the highest priority and system properties the lowest. Using this prioritization you can set and override default values.
4. Specify the server name.

   By default, the **launchClient** command uses *your_server_name* for the `BootstrapHost` property value.

   This setting is effective for testing your application client when it is installed on the same computer as the server. However, in other cases override this value with the name of your server. You can override the `BootstrapHost` value by invoking `launchClient` command with the following parameters:

   ```
   launchClient myapp.ear -CCBootstrapHost=abc.midwest.mycompany.com
   ```

   You can also override the default by specifying the value in a properties file and passing the file name to the launchClient shell.

   Security is controlled by the server. You do not need to configure security on the client because the client assumes that security is enabled. If server security is not enabled, then the server ignores the security request, and the application client functions as expected.

Example

You can store launchClient values in a properties file, which is a good method for distributing default values. You can then override one or more values on the command line. The format of the file is one `launchClient -CC` parameter per line without the `-CC` prefix. For example:

```
 verbose=true classpath=/usr/lpp/mydir/util.jar;/usr/lpp/mydir/harness.jar;/usr/lpp
/production/G19/global.jar BootstrapHost=abc.westcoast.mycompany.com tracefile=/usr
/lpp/WebSphere/mylog.txt
```

launchClient tool

This topic describes the Java Platform, Enterprise Edition (Java EE) command line syntax for the launchClient tool for WebSphere Application Server.

Note: All users who run commands from a specific profile must have authority to modify files that are created by other users that use the same profile. Otherwise, you might see a permission denied error in the log files. To avoid this issue, consider one of the following policies:

- Use specific profiles for distinct user authorities
- Always use the same user for all of the commands that are run in a given profile
- Ensure that all users of a specific profile belong to the same group. In addition, ensure that each user of a group has the read and write authority to the files that are created by other members in the same profile.

The following example illustrates the command line invocation syntax for the launchClient tool:

```
launchClient [-profileName pName | -JVMOptions options | -help | -?] <userapp> [-CC<name>=<value>] [app args]
```

where

- *userapp* is the path and the name of the EAR file that contains the application client.
- *-CC<name>=<value>* is the client container name-value pair parameter. See the client container parameters section, for supported name-value pair arguments.
- *app args* are arguments that pass to the application client.
- *-profileName* defines the profile of the Application Server process in a multi-profile installation. The -profileName option is not required for running in a single profile environment or in an Application Clients installation.
- *-JVMOptions* is a valid Java standard or nonstandard option string, except `-cp` or `-classpath`. Insert quotation marks around the string.
- *-help, -?* prints the usage information.

The first parameter must be `-help`, `-?` or contain no parameter at all. The `-profileName pName` and `-JVMOptions` options are optional parameters. If used, they must appear before the *<userapp>* parameter. All other parameters are optional and can appear in any order after the *<userapp>* parameter. The Java EE Application client run time ignores any optional parameters that do not begin with a `-CC` prefix and passes those parameters to the application client.

Client container parameters

Supported arguments include:

-CCadminConnectorHost
Specifies the host name of the server from which configuration information is retrieved.

The default is the value of the `-CCBootstrapHost` parameter or the value, `your.server.name`, if the `-CCBootstrapHost` parameter is not specified.

-CCadminConnectorPort
Indicates the port number for the administrative client function to use. The default value is `8880` for SOAP connections and `2809` for Remote Method Invocation (RMI) connections.

-CCadminConnectorType
Specifies how the administrative client connects to the server. Specify `RMI` to use the RMI connection type, or specify `SOAP` to use the SOAP connection type. The default value is `SOAP`.

-CCadminConnectorUser
Administrative clients use this user name when a server requires authentication. If the connection type is SOAP, and security is enabled on the server, this parameter is required.

-CCadminConnectorPassword
The password for the user name that the `-CCadminConnectorUser` parameter specifies.

-CCaltDD
The name of an alternate deployment descriptor file. This parameter is used with the `-CCjar` parameter to specify the deployment descriptor to use. Use this argument when a client JAR file is configured with more than one deployment descriptor. Set the value to `null` to use the client JAR file standard deployment descriptor.

-CCBootstrapHost
The name of the host server you want to connect to initially. The format is: *your_server_of_choice.com*

-CCBootstrapPort
The server port number. If you do not specify this argument, the WebSphere Application Server default value is `used`.

-CCclassLoaderMode
Specifies the class loader mode. If PARENT_LAST is specified, the class loader loads classes from the local class path before delegating the class loading to its parent. The classes loaded for the following are affected:

- Classes defined for the Java EE application client
- Resources defined in the Java EE application
- Classes specified on the manifest of the Java EE client JAR file
- Classes specified using the -CCclasspath option

If PARENT_LAST is not specified, then the default mode, PARENT_FIRST, causes the class loader to delegate the loading of classes to its parent class loader before attempting to load the class from its local class path.

-CCclasspath
A class path value. When you launch an application, the system class path is used. If you want to access classes that are not in the EAR file or part of the system class paths, specify the appropriate class path here. Multiple paths can be concatenated.

-CCD
Use this option to have the WebSphere Application Server set the specified system property during initialization. Do not use the equals (=) character after the `-CCD`. For example: `-CCDcom.ibm.test.property=testvalue`. You can specify multiple `-CCD` parameters. The general format of this parameter is `-CCD<property key>=<property value>`. For example, `-CCDI18NService.enable=true`.

-CCdumpJavaNameSpace
Controls generation of a dump of the java: name space for the application that is launched, which can be used for debugging purposes. A value of **true** generates a dump in short format, and includes the name and object type for each binding. A value of **long** generates a dump in long format, and includes additional information for each binding over short format, such as the local object type and string representation of the local object. The default value is **false**, and does not generate a dump.

-CCexitVM
Use this option to have the WebSphere Application Server call the `System.exit()` method after the client application completes. The default is `false`.

-CCinitonly
Use this option to initialize application client run time for ActiveX application clients without launching the client application. The default is `false`.

-CCjar
The name of the client Java Archive (JAR) file that resides within the EAR file for the application you wish to launch. Use this argument when you have multiple client JAR files in the EAR file.

-CCpropfile
Indicates the name of a properties file that contains launchClient properties. Specify the properties without the `-CC` prefix in the file, with the exception of the securityManager, securityMgrClass and securityMgrPolicy properties. See the following example: `verbose=true`.

-CCproviderURL
Provides bootstrap server information that the initial context factory can use to obtain an initial context. WebSphere Application Server initial context factory can use either a Common Object Request Broker Architecture (CORBA) object URL or an Internet Inter-ORB Protocol (IIOP) URL. CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. This value can contain more than one bootstrap server address. This feature can be used when attempting to obtain an initial context from a server cluster. You can specify bootstrap server addresses, for all servers in the cluster, in the URL. The operation will succeed if at least one of the servers is running, eliminating a single point of failure. The address list does not process in a particular order. For naming operations, this value overrides the `-CCBootstrapHost` and `-CCBootstrapPort` parameters. A CORBA object URL specifying multiple systems is illustrated in the following example:

```
-CCproviderURL=corbaloc:iiop:myserver.mycompany.com:9810,:mybackupserver.mycompany.com:2809
```

This value is mapped to the `java.naming.provider.url` system property.

-CCsecurityManager
Enables and runs the WebSphere Application Server with a security manager. The default is `disable`.

-CCsecurityMgrClass
Indicates the fully qualified name of a class that implements a security manager. Only use this argument if the `-CCsecurityManager` parameter is set to `enable`. The default is `java.lang.SecurityManager`.

-CCsecurityMgrPolicy
Indicates the name of a security manager policy file. Only use this argument if the `-CCsecurityManager` parameter is set to enable. When you enable this parameter, the `java.security.policy` system property is set. The default is *<app_server_root>*/ `properties/client.policy`.

-CCsoapConnectorPort
The Simple Object Access Protocol (SOAP) connector port. If you do not specify this argument, the WebSphere Application Server default value is used.

-CCtrace
Use this option to obtain debug trace information. You might need this information when reporting a problem to IBM customer support. The default is `false`. For more information, read the topic ″Enabling trace.″

-CCtracefile
Indicates the name of the file to which trace information is written. The default is to write output to the console.

-CCtraceMode
Specifies the trace format to use for tracing. If the valid value, `basic`, is not specified the default is advanced. Basic tracing format is a more compact form of tracing.

For more information on basic and advanced trace formatting, see Interpreting trace output.

-CCverbose
This option displays additional information messages. The default is `false`.

If you are using an EJB client application with security enabled, edit the `sas.client.props` file, which is located in the *profile_root*`/properties` directory. Within the file, change the `com.ibm.CORBA.loginSource` value to *none*.

For more information on the `sas.client.props` utility, see Manually encoding passwords in properties files and the PropFilePasswordEncoder command reference.

RMI connection with security. Used with the EJB and administrative client application.

Using Jacl:

```
wsadmin -conntype RMI -port  rmiportnumber -user userid
-password password
```

Using Jython:

```
wsadmin -lang jython -conntype RMI -port  rmiportnumber -user userid
-password password
```

rmiportnumber for your connection displays in the administrative console as BOOTSTRAP_ADDRESS.

Note: On the AIX, HP-UX, Linux, i5/OS, Solaris, and z/OS operating systems, the use of -password option may result in security exposure as the password information becomes visible to the system status program, such as ps command, which can be invoked by other users to display all of the running processes. Do not use this option if security exposure is a concern. Instead, specify user and password information in the `soap.client.props` file for SOAP connector or `sas.client.props` file for RMI connector. The `soap.client.props` and `sas.client.props` files are located in the properties directory of your WebSphere Application Server profile.

If Kerberos (KRB5) is enabled for administrative authentication, the authentication target supports BasicAuth and KRB5. To use KRB5, update the `sas.client.props`, `soap.client.props`, and `ipc.client.props` files, according to the connector type.

Note: When using Kerberos authentication, the user password does not flow across the wire. A one-way hash of password is used to identify the client.

The following examples demonstrate correct syntax.

/QIBM/ProdData/WebSphere/AppServer/V61/Base/bin/launchClient /home/earfiles/myapp.ear -profileName myprofile -CCBootstrapHost=myWASServer -CCverbose=true app_parm1 app_parm2

Specifying the directory for an expanded EAR file

You can archive the `Manifest.mf` client Java Archive (JAR) files instead of automatically cleaning them up after the application exits.

Before you begin

Each time the launchClient tool is called, it extracts the Enterprise Archive (EAR) file to a random directory name in the temporary directory on your hard drive. Then the tool sets up the thread ClassLoader to use the extracted EAR file directory and JAR files included in the `Manifest.mf` client Java Archive (JAR) file. In a normal J2EE Java client, these files are automatically cleaned up after the application exits. This cleanup occurs when the client container shutdown hook is called. To avoid extracting the EAR file (and removing the temporary directory) each time the launchClient tool is called, complete the following steps:

1. Specify a directory to extract the EAR file by setting the `com.ibm.websphere.client.applicationclient.archivedir` Java system property. If the directory does not exist or is empty, the EAR file is extracted normally. If the EAR file was previously extracted, the launchClient tool reuses the directory.
2. Delete the directory before running the launchClient tool again, if you need to update your EAR file. When you call the `launchClient` command, it extracts the new EAR file to the directory. If you do not delete the directory or change the system property value to point to a different directory, the launchClient tool reuses the currently extracted EAR file and does not use your changed EAR file. When specifying the `com.ibm.websphere.client.applicationclient.archivedir` property, make sure that the directory you specify is unique for each EAR file you use. For example, do not point the `MyEar1.ear` and the `MyEar2.ear` files to the same directory.

Using Java Web Start

Learn about the Java Web Start technology that is provided by the Java Standard Edition runtime environment to deploy Java Enterprise Edition application clients, including Thin application clients, on the remote client machine with a single click from Web browser on the client machine.

Before you begin

The supported client platforms for deploying application clients using the Java Web Start are the same as the IBM Application Client for WebSphere Application Server supported platforms, except Linux on Power and OS/400 operating systems.

Before you begin this task, see the following topics to understand Java Web Start technology and its components:

- "Java Web Start architecture for deploying application clients" on page 210
- "Client application Java Network Launcher Protocol deployment descriptor file" on page 211
- "ClientLauncher class" on page 215

You can use the following resources:

- The IBM implementation of Java Web Start on Java Platform Standard Edition 6 (Java SE 6) that is packaged in the Application Client for WebSphere Application Server.

Note: When using the Sun JRE with the Application Client runtime, application resources cannot be downloaded using the https protocol. This is a Sun Microsystems limitation. Install the IBM JRE on the client machine and use IBM Java Web Start to run the WebSphere Application Client application. Sun Java Web Start is supported, but with a limitation that only the http protocol can be used to download the application resource, including the application jnlp description file.

About this task

To deploy application clients using Java Web Start, the client machine must have at least a Java SE runtime environment installed. The Java SE runtime environment includes the Java Web Start, which implements the JSR 56: Java Network Launching Protocol and API. The application clients Enterprise Archive (EAR) file is a Java archive (JAR) resource in a JNLP descriptor file that resides on a central server. The JNLP descriptor file also specifies the runtime environment requirement for running the application.

WebSphere Application Server provides a launcher class to launch the Java EE application client in the application client container inside of Java Web Start. The client machine might not have the IBM Application Client for WebSphere Application Server installed. If this is the case, create and install an application client container and runtime package as a runtime environment through Java Web Start. The JNLP descriptor file specifies this runtime environment as the required runtime environment for running the Java EE application client.

WebSphere Application Server also provides command-line utility programs to create this application client container and runtime package from an existing IBM Application Client for WebSphere Application Server installation, as well as an installer class to install this package as a runtime environment for the application client container and also the Java Runtime Environment (JRE) in the IBM Application Client for WebSphere Application Server installation. To run the Java EE application client, the EAR file is deployed as a JAR resource that is described in the JNLP descriptor file.

1. Identify the client machine operating system, and install the corresponding IBM Application Client for WebSphere Application Server on a development machine. For example, if the Java EE application clients are targeted to run on Windows operating systems, install the IBM Application Client for WebSphere Application Server for Windows. Follow the instructions for "Installing Application Client for WebSphere Application Server" on page 242.

2. Run the utility programs to create the application client container and runtime package.
   a. Use the "buildClientRuntime tool" on page 218 utility to create the package.
   b. Use the "buildClientLibJars tool" on page 211 utility to create the JAR files containing the launcher and the installer class. This utility also zips up the properties files in the <app_client_root>/ properties directory.
3. Create the runtime installer JNLP descriptor file. The JNLP response must be included in the JNLP version ID to indicate the current runtime version in the response header, for example, `x-java-jnlp-version-id=1.6.0`. Using a servlet of a Java Server Pages (JSP) file to provide a dynamic JNLP response.
4. Create the Java EE application client launch JNLP descriptor file.
5. Package the application client container runtime environments and the Java EE application in an Enterprise Archive (EAR) file. Depending on your preferred deployment strategy, the files can be in two separate Web modules, or combined into one.
6. All JAR resources must be Java signed, including the Java EE application client EAR file.
7. Deploy the Enterprise Archive file on an application server, and start the application. The Java EE application client is ready to be deployed.

Example

A Java Web Start deployment Sample is included in the client samples. This Sample demonstrates the steps to deploy a Java EE application client with an automated ANT script. The Sample has a servlet to generate the runtime installer JNLP response with JNLP version ID, for example, `x-java-jnlp-version-id`.

Note: When the application client initially launches using Java Web Start from Sun Microsystems Java SE Runtime Environment 6.0, it installs the Application Client runtime, which includes the IBM JRE. An null pointer exception (NPE) is thrown from the `com.sun.deploy.services.WPlatformService.getSecureRandom()` method. This is a known bug in Sun Java SE 6 (http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6505528). If you experience this exception, relaunch the application. The NPE only occurs on the first launch of the application client.

Java Web Start architecture for deploying application clients

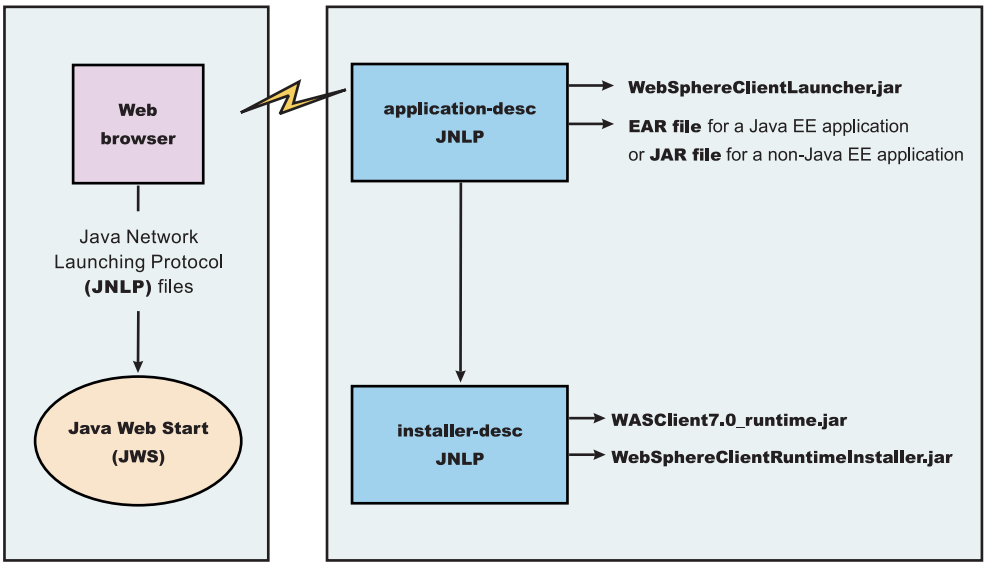
Java Web Start is an application-deployment technology that includes the portability of applets, the maintainability of servlets and JavaServer Pages (JSP) file technology, and the simplicity of mark-up languages such as XML and HTML. It is a Java application that allows full-featured Java EE client applications to be launched, deployed and updated from a standard Web server. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported.

Upon launching Java Web Start for the first time, you might download new client applications from the Web. Each time you launch JWS thereafter, you can initiate applications either through a link on a Web page or (in Windows) from desktop icons or the Start menu. You can deploy applications quickly using Java Web Start, cache applications on the client machine, and launch applications remotely offline. Additionally, because Java Web Start is built from the Java Platform, Enterprise Edition (Java EE) infrastructure, the technology inherits the complete security architecture of the Java EE platform.

The technology underlying Java Web Start is the Java Network Launching Protocol & API (JNLP). Java Web Start is a JNLP client and it reads and parses a JNLP descriptor file (JNLP file). Based on the JNLP descriptor, it downloads appropriate pieces of a client application and any of its dependencies. If any of the pieces of the application are already cached on the client machine, then those components are not downloaded again, unless they have been updated on the server machine. After you download and cache the client application, JWS launches it natively on the client machine.

The following diagram shows an overview of launching a client application, include the Application Client for WebSphere Application Server as a dependent resource, using Java Web Start.



The Web browser running on a client machine connects to a Web application located on a server machine. The client application JNLP descriptor file is downloaded and processed by Java Web Start on the client machine.

In this diagram, there are two JNLP descriptor files:

- Client application JNLP descriptor (application-desc in the diagram)
- Application Clients run-time installer JNLP descriptor (installer-desc in the diagram)

Each of these JNLP descriptor files, the client application (JAR or EAR) and the dependent resource JAR files are packaged as Web applications in an EAR file. This EAR file is deployed to an Application server. The client machine with JWS installed uses a Web browser to connect to the url of the client application JNLP descriptor file to download and run the client application.

Using Java Web Start from Java SE Runtime Environment 6.0 or later is highly recommended. All the platforms supported by the application client for WebSphere Application Server are supported with the exception Linux on Power and OS/400 platforms.

You can use the following:

- Java Web Start on the Java Standard Edition Developer Kits that IBM provides, packaged in Application Client for WebSphere Application Server
- Java Web Start on Sun Microsystems Java SE 6 Development Kit or Java SE Runtime Environment 6.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems
- Java Web Start on HP-UX JDK or JRE for Java Platform, Standard Edition, version 6, which you can download from the HP Web site

buildClientLibJars tool:

For a Java Platform, Enterprise Edition (Java EE) application client application and or Thin application client application to be launched using Java Web Start (JWS), the properties files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to create those property JAR files. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported on WebSphere Application Server for i5/OS.

The buildClientLibJars tool copies the JAR files from the Application Client for WebSphere Application Server installation and creates a `properties.jar` file, which contains the properties files from the Application Clients installation properties directory to a specified location. When this property is created, the tool uses the value of `keystore`, `storepass`, `alias` and `storetype` to sign all of the JAR files in the specified location.

```
Windows usage: buildClientLibJars.bat  [-help] [-verbose] destdir keystore storepass alias storetype

Unix usage:    buildClientLibJars.sh  [-help] [-verbose] destdir keystore storepass alias storetype
```

where:

- `-help` will display the message
- `-verbose` will turn on verbose message
- `destdir` will output the destination directory name
- `keystore` is the key store file
- `storepass` is the key store password
- `alias` key is the alias name
- `storetype` is the key store type

Client application Java Network Launcher Protocol deployment descriptor file

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Location

The client application has an Application Clients run-time dependency that provides the following:

- Java SE Runtime Environment from IBM
- Application Clients run-time properties
- SSL KeyStore and TrustStore file
- Application Clients run-time library JAR files (optional for Thin Application client applications)

If the Application Clients run-time dependency is not met, it is downloaded and installed in Java Web Start (JWS), as described by the Application Clients run-time installer JNLP descriptor file. For example:

```
<j2se version="1.6" href="http://your_server.com/jws/wasappclient/download.jnlp"/>
```

Usage notes

The client application must also include the `WebSphereClientLauncher.jar` file, which contains the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, that completes one of the following actions:

- If it is a Java Platform, Enterprise Edition (Java EE) Application client application (that is the resources for the application contain an EAR file with a client application), the EAR file must be specified as a JAR resource so that it can be downloaded to JWS and specified in the system property, `com.ibm.websphere.client.launcher.ear`. See “JNLP descriptor file for a Java EE Application client application” on page 213 for an example.
- If it is a Thin Application client application, the Thin Application client application JAR file must be specified as a JAR resource so that it can be downloaded to JWS and the name of the class containing main method entry point is specified in the system property, `com.ibm.websphere.launcher.main`. See “JNLP descriptor file for a Thin Application client application” on page 213 for an example.

The JNLP specification requires all the resource (JAR or EAR) files used in a JNLP file to be signed.

You can specify the –CC arguments defined in the launchClient tool for a J2EE Application client application in application arguments section of the JNLP descriptor files. However, only –CCD is supported for a Thin Application client application to define system properties and the JNLP <property> tag can also be used to define system properties. See the following example for details:

```
<property name="java.naming.provider.url" value="corbaloc:iiop:myserver.com:9089"/>
```

For a J2EE Application client application, specify the following application arguments as defined in the JNLP.

1. Specify your target server provider URL, as shown in the following example:

   ```
   <argument> >-CCDjava.naming.provider.url =corbaloc:iiop:myserver.mydomain.com:9080 </argument>
   ```

2. Specify the SSL Key File and SSL Trust File location. These files are expected to be available in the client machine. To use the ones in the Application Clients run-time dependency installed in JWS cache, specify these application arguments:

   ```
   <argument> -CCDcom.ibm.ssl.keyStore=${WAS_ROOT}/etc/key.p12 </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=${WAS_ROOT}/etc/trust.p12 </argument>
   ```

3. Specify the initial naming context factor, as shown in the following example:

   ```
   <argument>-CCDjava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory </argument>
   ```

   For a Thin Application client application, you also need to specify the actual location of the `sas.client.props` and `ssl.client.props` files located in the Application Clients run-time dependency that is installed in the JWS cache.

   ```
   <argument>-CCDcom.ibm.CORBA.ConfigURL=file:${WAS_ROOT}/properties/sas.client.props </argument>
   <argument>-CCDcom.ibm.SSL.ConfigURL=file:${WAS_ROOT}/properties/ssl.client.props </argument>
   ```

   If any of the default settings in the `sas.client.props` and `ssl.client.props` file need modifying, use the –CCD to change the settings through the system properties, as shown in the following example:

   ```
   <argument>-CCDjavacom.ibm.CORBA.securityEnabled=false </argument>
   ```

   Note: The ${WAS_ROOT} token used in the JNLP file is replaced by the launcher class, com.ibm.websphere.client.launcher.ClientLauncher, to the actual location of the Application Clients run-time dependency installation in the JWS cache. If you are using JSP to dynamically create this JNLP description file, you must escape this token because it has a different meaning in JSP 2.0. See the following example for details:

   ```
   <argument>-CCDcom.ibm.ssl.keyStore=\${WAS_ROOT}/etc/key.p12 </argument>
   <argument>-CCDcom.ibm.ssl.trustStore=\${WAS_ROOT}/etc/trust.p12 </argument>
   ```

JNLP descriptor file for a Java EE Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application.

Here is an example of the client application JNLP descriptor file for a Java EE Application client application:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
```

```
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->

<jnlp spec="1.0+" codebase="http://your_server:port_number/jws/wasappclient/apps/">
 <information>
  <title>Java EE Client Example</title>
  <vendor>IBM</vendor>
  <homepage href="null"/>
  <description>Java WebStart example: Launching Java EE Application Client</description>
  <description kind="short">Java EE Applicaiton Client</description>
  <description kind="tooltip">Java EE Application Client</description>
 </information>

 <security>
  <all-permissions/>
 </security>

 <resources>
  <j2se href="http://your_server:port_number/jws/wasappclient/JREDownload.xjnlp" version="1.6"/>
  <jar href="../lib/WebSphereClientLauncher.jar" download="eager" main="false"/>
  <jar href="../lib/properties.jar" download="eager" main="false"/>
  <jar href="SwingCalculator.ear" download="eager" main="false"/>

  <property name="com.ibm.websphere.client.launcher.ear" value="SwingCalculator.ear"/>
 </resources>

 <application-desc main-class="com.ibm.websphere.client.launcher.ClientLauncher">
  <argument>-CCproviderURL=corbaloc:iiop:tiu03.torolab.ibm.com:2809</argument>
 </application-desc>

</jnlp>
```

JNLP descriptor file for a Thin Application client application:

The deployment descriptor file is the main Java Network Launcher Protocol (JNLP) descriptor file for the client application. If it is a Thin Application client application, then the launcher class uses the current JVM from the Application Clients run-time dependency and invokes the Thin Application client application main method.

Here is an example of the JNLP descriptor file for a Thin Application client application.

```
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer (a) for its own instruction and study, (b) in order
to develop applications designed to run with an IBM WebSphere product, either for customer's
own internal use or for redistribution by customer, as part of such an application, in
customer's own products.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.

Licensed Materials - Property of IBM

5724-I63, 5724-H88, 5724-H89, 5655-N02, 5724-J08

Copyright IBM Corp. 2008 All Rights Reserved.

US Government Users Restricted Rights - Use, duplication or
```

```
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
-->

<!-
=====================================================================
->
<!-- TODO: change "codebase" to the actual URL location of the jnlp file ->
=====================================================================
->
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://your_server:port_number/jws/wasappclient/apps">
  <information>
  <title>Thin Base Calculator Client Samples</title>
  <vendor>IBM</vendor>
  <description>Thin Base Calculator Client Samples</description>
  <offline-allowed/>
 </information>

 <security>
  <all-permissions/>
 </security>
  <resources>
   <j2se version="1.6" href="http://your_server:port_number/jws/wasappclient/JREDownload.xjnlp"/>

   <jar href="/jws/wasappclient/lib/WebSphereClientLauncher.jar" main="true"/>
   <jar href="BasicCalculatorClientCommon.jar"/>
   <jar href="BasicCalculatorEJB.jar"/>
   <jar href="BasicCalculatorThinClient.jar"/>

   <property name="com.ibm.websphere.client.launcher.main"
      value="com.ibm.websphere.samples.technologysamples.basiccalcthinclient.BasicCalculatorClientThinMain"/>
   <property name="java.naming.factory.initial"
              value="com.ibm.websphere.naming.WsnInitialContextFactory" />
   <property name="java.naming.provider.url"
               value="corbaloc:iiop:tiu03:2809"/>

   </resources>

   <add</argument>
    <argument>1</argument>
    <argument>2</argument>
   </application-desc>
</jnlp>
```

ClientLauncher class:

The class, com.ibm.websphere.client.installer.ClientLauncher, contains a main() method that is called by Java Web Start (JWS) to launch the client application. The Java Web Start client is used with platforms that support a Web browser.

Java Web Start is not supported.

This client is packaged in the `WebSphereClientLauncher.jar` file that is located in the Application Client for WebSphere Application Server installation under the *<app_client_root>*/ lib/webstart directory.

The launcher class requires that the following properties are defined. These properties are not defined in a separate properties file. Instead, the properties are defined as part of the Java Network Launching Protocol (JNLP) files.

com.ibm.websphere.client.launcher.main
: If the client application is a Thin Application client, then this property should be specified. It specifies the class where the main entry point of the client application resides.

com.ibm.websphere.client.launcher.ear
If the client application is a Java Platform, Enterprise Edition (Java EE) Application client, then this property should be specified. It specifies the name of the EAR file to be executed. This property takes precedence over com.ibm.websphere.client.launcher.main. However, only one of the two properties should be specified.

Application client launcher for Java Web Start:

The application client launcher for Java Web Start is a Java class, com.ibm.websphere.client.installer.ClientLauncher, which has a main() method that Java Web Start calls to start the application client container and to invoke the application client's main() method. It provides similar functions as the **lauchClient** command line tool to start application clients from the command line.

The com.ibm.websphere.client.launcher.ClientLauncher class is packaged in the WebSphereClientLauncher.jar file under the <app_client_root>/lib/webstart directory.

The launcher tool requires that the following properties are defined.

com.ibm.websphere.client.launcher.main
If the client that is to be run is a thin client, then this property should be specified. It specifies the class where the main entry point of the application resides. It is the main class name for a Thin application client. If it is set, the launcher will not start the client container, it will rather invoke the main method for the application directly. However, if `com.ibm.websphere.client.launcher.ear` is also set, it will be ignored.

com.ibm.websphere.client.launcher.ear
If the client that is to run is the Java Platform, Enterprise Edition (Java EE) client, then this property should be specified. It specifies the name of the ear file to be executed. This property takes precedence over `com.ibm.websphere.client.launcher.main` although only one of the two properties should be specified.

These properties are not defined in a separate properties file. Instead, they are defined as part of the Java Network Launching Protocol files.

When `com.ibm.websphere.client.launcher.ear` is set, the application client launcher for JWS supports almost all of the –CC arguments as the **lauchClient** command line tool supports. However, if only `com.ibm.websphere.client.launcher.main` is set, the launcher will only support the –CCD argument. The following table shows the comparison of the supported –CC arguments for the **launchClient** command line tool and the application client launcher for JWS:

-CC argument	launchClient	Application client launcher for JWS
-CCverbose	Yes	Yes
-CCjar	Yes	Yes
-CCclasspath	Yes	N/A
-CCadminConnectorHost	Yes	Yes
-CCadminConnectorPort	Yes	Yes
-CCadminConnectorType	Yes	Yes
-CCadminConnectorUser	Yes	Yes
-CCaltDD	Yes	Yes
-CCBootstrapHost	Yes	Yes
-CCBootstrapPort	Yes	Yes
-CCproviderURL	Yes	Yes
-CCinitonly	Yes	N/A

-CC argument	launchClient	Application client launcher for JWS
-CCtrace	Yes	Yes
-CCtracefile	Yes	Yes
-CCsecurityManager	Yes	N/A
-CCsecurityMgrClass	Yes	N/A
-CCsecurityMgrPolicy	Yes	N/A
-CCD	Yes	Yes
-CCexitVM	Yes	Yes
-CCdumpJavaNameSpace	Yes	Yes
-CCsoapConnectorPort	Yes	Yes
-CCtraceMode	Yes	Yes
-CCclassLoaderMode	Yes	Yes

Macro expansion is supported for the –CCD argument by the application client launcher for JWS. The launcher will automatically substitute certain macro keys (enclosed with ${...}) with the calculated value at runtime. For example, if a macro key is used in the –CCD argument in the application client JNLP manifest file,

```
<argument>-CCDcom.ibm.ssl.keyStore= ${WAS_ROOT}/etc/key.p12</argument>
```

it will be expanded to the JWS cache installation root location and the argument will become:

```
-CCDcom.ibm.ssl.keyStore=/home/tiu/.java/deployment/cache/javaws/ext/E1134532441112/etc/key12.p12
```

The following table shows the 3 macro keys that are currently supported and will be substituted by the launcher:

Table 3.

Macro key	Value
${WAS_ROOT}	Installation root location within the JWS cache that is used by the application client container and runtime installer for JWS.
${JAVA_HOME}	Location of Java home. The return value of System.getProperty(“java.home”).
${USER_HOME}	Location of user home. The return value of System.getProperty(“user.home”).

Preparing the application client run time dependency component for Java Web Start

To launch a Java Platform, Enterprise Edition (Java EE) application client application, a Thin application client application, or both using Java Web Start (JWS), a Java Runtime Environment implementation Java archive (JAR) that IBM provides, the library JAR files and properties files bundled in Application Client for WebSphere Application Server must be installed in the JWS. Learn the steps to build the application client run time dependency component from an application client installation. It is packaged as a Web Archive Resource (WAR) file that can be installed in an Application Server.

Before you begin

Install the Application Client for WebSphere Application Server for the operating system to which the client application deploys. If there is a requirement to deploy the client application to multiple operating systems,

the application client run time dependency component must be built separately for each operating system that client application supports.

1. Install the Application Client for WebSphere Application Server for the client application supported operating systems.
2. Change the directory to the installation bin directory.
3. Run the “buildClientRuntime tool” on page 218 to generate the application client run time JAR file, which contains the Java Standard Edition Runtime Environment, the run time library JAR files, properties files, and the SSL KeyStore and TrustStore files from the application client installation.
4. Run the buildClientLibJars tools to package up the properties files in the properties directory of the application client installation into a properties.jar file in the specified location. The buildClientLibJars tools will also copy the `WebSphereClientLauncher.jar` file and `WebSphereClientRuntimeInstaller.jar` file from the application client installation to the specified location. All jar files in the specified location will be signed by the provided certificate.

   For example, if you are using Version 7.0 and using the test certificate that is included in the application client installation:

   ```
   buildClientLibJars C:\Temp\webstart ..\etc\DummyClientKeyFilejar WebAS "websphere dummy client" JKS
   ```
5. Create a JavaServer Pages (JSP) file or use a servlet to generate the application client run time installer Java Network Launching Protocol (JNLP) descriptor to respond to Java Web Start request. See the Java Web Start Deployment sample in the application client installation.
6. Package the two signed JAR files, `WASClient7.0_windows.jar` and `WebSphereClientRuntimeInstaller.jar`, and the JSP file or servlet for generating the Application Client run time installer JNLP descriptor into a Web archive (WAR) file. This WAR file is packaged into an EAR file that can be deployed to an application server. See the Java Web Start Deployment sample in the application client installation.

Results

Your Web application is ready to serve the application client run time and the JRE environment.

buildClientRuntime tool:

For a Java Platform, Enterprise Edition (Java EE) application client application and or Thin application client application to be launched using Java Web Start (JWS), the library JAR files bundled in Application Client for WebSphere Application Server must be installed in the Java Web Start. Use this tool to build those JAR files. The Java Web Start client is used with platforms that support a Web browser.

The Java Web Start client is not supported on WebSphere Application Server for OS/400.

The buildClientRuntime tool builds the required components from the WebSphere Application Server clients installation into the JAR file specified on the command. This JAR file contains:

- License files
- Java SE Runtime Environment 6 (JRE 6) that IBM provides
- Application Clients runtime properties and configuration
- SSL KeyStore and TrustStore files
- Runtime library JAR files

In the case of building an Application Clients runtime JAR file only for serving Thin Application client applications and not for Java EE Application client applications, the runtime library JAR files and the Application Clients runtime properties files are not included, except the configuration files, `sas.client.props`, `ssl.client.props` and `soap.client.props`, located in the `WAS_ROOT/properties` directory. The Java Web Start client is used with platforms that support a Web browser.

The command-line invocation syntax for the buildClientRuntime tool is shown in the following example:

```
Windows Usage:    buildClientRuntime.bat  [-help] [-verbose] outfile keystore storepass alias storetype

Unix Usage:         buildClientRuntime.sh    [-help] [-verbose] outfile keystore storepass alias storetype
```

where:

- -help will display the message
- -verbose will turn on verbose message
- outfile is the output file name
- keystore is the key store file
- storepass is the key store password
- alias is the key alias name
- storetype is the key store type

ClientRuntimeInstaller class:

This section provides information on the ClientRuntimeInstaller class.

This class, com.ibm.websphere.client.installer.ClientRuntimeInstaller, contains a main() method that Java Web Start (JWS) calls to install the Application Client for WebSphere Application Server run-time dependency component in JWS cache. It is packaged in `WebSphereClientRuntimeInstaller.jar` file located in the Application Client for WebSphere Application Server installation in the *<app_server_root>*/`JWS` directory.

Specify the `WebSphereClientRuntimeInstaller.jar` file and the Application Client run-time dependency component JAR file as JAR resources in the Application Client run-time installer Java Network Launcher Protocol (JNLP) descriptor file. See the following example for details:

```
<jar href="Launcher/WebSphereClientRuntimeInstall.jar" main="true"/>
<jar href="Launcher/WASClient6.1_windows.jarRuntimeInstall.jar" main="true"/>
```

The ClientRuntimeInstaller class main method requires the following properties to be set in the JNLP file:

com.ibm.websphere.client.jre.version
: Specifies a Java Runtime Environment (JRE) version name that is to be used when referring to the Application Client run-time dependency component.

com.ibm.websphere.client.jre.launch.java
: Specifies the relative location of the javaw.exe program in the Application Client run-time dependency component JAR file.

The previously mentioned properties, JRE version name and the location of the javaw.exe program are registered to the Java Web Start Application Manager, as shown in the following example:

```
<property name="com.ibm.websphere.client.jre.version" value="WASclient6.1"/>
<property name="com.ibm.websphere.client.jre.launch.java" value="java\jre\bin\javaw.exe"/>
```

Using the Java Web Start sample

The EAR file, `WebSphereClientRuntime.ear`, is provided in the JWS directory of the Client Application for WebSphere Application Server installation. This EAR file provides a sample Application Clients run-time installer JNLP descriptor file and a sample Application Clients run-time library component JNLP descriptor file. Follow the steps in this task to build the Application Clients run-time dependency component and the Application Clients run-time library component. Add these components to the `WebSphereClientRuntime.ear` file, and then install the EAR file in an Application Server to be used by the client application.

About this task

There is a new Java Web Start sample available in the client sample gallery for WebSphere Application Server V7.0. Refer to the client sample gallery in the Application Client for WebSphere Application Server

product. The name of the new sample is ″Java Web Start Deployment Sample″.

Installing Java Web Start

Learn about the steps that are necessary to install Java Web Start (JWS) on the AIX platform. Java Web Start technology is provided by the Java SE runtime environment to deploy Java EE application clients (including Thin application clients) on the remote client machine with a single click from Web browser on the client machine.

Before you begin

Note: This topic applies only to the AIX operating system.

Before you begin this task, see the “Preparing the application client run time dependency component for Java Web Start” on page 217 topic to understand Java Web Start (JWS) technology and components.

Note:

You can use the following resources:

- The IBM implementation of Java Web Start on Java Platform Standard Edition 6 (Java SE 6) that is packaged in the Application Client for WebSphere Application Server

Note: You can use Java Web Start on Java Platform, Standard Edition Developer Kit 6 that IBM provides, packaged in the Application Client for WebSphere Application Server, Version 7.0; Java Web Start on Sun Microsystems Java Standard Edition Software Development Kit 6 or Java SE Runtime Environment 6.0, which you can download from the Sun Microsystems Web site for Windows, Linux and Solaris operating systems, or the Java Web Start on HP Software Development Kit (SDK) or RTE for Java 2 version 5.0, which you can download from the HP Web site.

About this task

Use the following steps to install JWS on the AIX platform.

1. Install IBM Application Client for WebSphere Application Server.
2. Change your directory to the client_install_root/java/jre/lib/javaws path.
3. Run the updateSetting.sh script from the path mentioned in the previous step.
4. Change your path to the JWS installed path. For example, enter: `client_install_root/java/jre/javaws`.
5. Run `./javaws` from the path mentioned in the previous step.

Using a static JNLP file with Java Web Start for Application clients

Do not use JSP to dynamically generate a JNLP file, otherwise the JNLP jsp page cannot be opened in some IE browsers.

About this task

To use a static JNLP file, you will need to add the following mime type mapping in the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
       <display-name>
       WAS Client runtime for Java Web Start</display-name>
       <welcome-file-list>
              <welcome-file>index.html</welcome-file>
              <welcome-file>index.htm</welcome-file>
              <welcome-file>index.jsp</welcome-file>
```

```
            <welcome-file>default.html</welcome-file>
            <welcome-file>default.htm</welcome-file>
            <welcome-file>default.jsp</welcome-file>
        </welcome-file-list>
        <mime-mapping>
        <extension>jnlp</extension>
        <mime-type>application/x-java-jnlp-file</mime-type>
        </mime-mapping>
</web-app>
```

Running the IBM Thin Client for Enterprise JavaBeans (EJB)

An EJB Client is a Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) Java Platform, Standard Edition (Java SE) application that accesses remote Enterprise Java Beans from a server through Java Naming and Directory Interface (JNDI) look up. IBM Thin Client for EJB offers a smaller footprint and is easy to deploy to a Java SE environment and an Eclipse Rich Client Platform (RCP) environment. You can bundle the IBM Thin Client for EJB library using the WebSphere Application Server installation or the Application Client for WebSphere Application Server installation with your application. The IBM Thin Client for EJB also extends the choice of Java SE runtime. It can be run in the Java Runtime Environment (JRE) that is packaged with the WebSphere Application Server product, the Sun Microsystems JRE that is downloaded from the Sun Microsystems Web site, or the JRE that is downloaded from the HP Web site.

Before you begin

The IBM ORB implementation library is required if the IBM Thin Client for EJB is running with a non-IBM product JRE on a non-IBM product platform. For example, running the IBM Thin Client for EJB with Sun Microsystems JRE on Windows, Linux, or Solaris, and with the HP JRE on HPUX. The IBM-provided Solaris hybrid and HP hybrid JRE are not considered non-IBM product JRE environments. The IBM Thin Client for EJB can access version 2.x and version 3.0 EJB on the WebSphere Application Server using the JNDI lookup, but it cannot access version 3.0 EJB through resource injection. Resource injection is supported if the client application is a Java Platform, Enterprise Edition (Java EE) Application Client running within the Java Platform, Enterprise Edition (Java EE) Application Client Container.

Before you set up an EJB Thin Client environment, obtain the Java archive (JAR) file for the EJB Thin Client for WebSphere Application Server. To obtain the EJB Thin Client for WebSphere Application Server, install WebSphere Application Server or Application Client. The EJB Thin Client for WebSphere Application Server file, `com.ibm.ws.ejb.thinclient_7.0.0.jar`, is located in the *app_server_root*`\runtimes` directory.

Copy the Java archive (JAR) file for the IBM Thin Client for EJB with WebSphere Application Server product, `com.ibm.ws.ejb.thinclient_7.0.0.jar` , to other machines to create a lightweight client environment that enables communications with the products. Copies of the IBM Thin Client for EJB are subject to the same terms and conditions of the license agreement for Version 7.0 WebSphere Application Server where you obtained the Thin Client for EJB. Refer to the license agreements for correct usage and other limitations.

The IBM Thin Client for EJB with WebSphere Application Server runs on distributed operating systems with JDK support, including both Version 5 and Version 6. When using the IBM Thin Client for EJB as a standalone Java SE application with a non-IBM product JRE, you must override the default ORB implementation for the JRE through one of following methods:

- Include the com.ibm.ws.orb_7.0.0.jar file in the Java system classpath.
- Override the default ORB implementation in the JRE, using Java Endorsed Standards Override Mechanism.
- Set the java.endorsed.dirs path to a directory that contains the com.ibm.ws.orb_7.0.0.jar file.

When running the IBM Thin Client for EJB as an Eclipse RCP application, it is recommended to use method two, to override the default JRE ORB implementation.

Note: The Pluggable Application Client feature in the application client installer is deprecated. It is replaced by the IBM Thin Client for EJB.

About this task

Run the IBM Thin Client for EJB, by completing the following steps.

1. Invoke the client application. Run the following Java command:

   Add the following system properties to the Java command if you want authentication and SSL enabled:

2. Provide IIOP authentication configuration and Client SSL Configuration. Add the following system properties to the Java command:

   ```
   -Dcom.ibm.SSL.ConfigURL=file:///home/user1/ssl.client.props
   -Dcom.ibm.CORBA.ConfigURL=file:///home/user1/sas.client.props
   ```

   You can obtain the `ssl.client.props` file and `sas.client.props` file from the WebSphere Application Server installation and modify the file to suit your environment. You must, at a minimum, update the location of the key files in the `ssl.client.props` file to the match location of your target environment. For example,

   ```
   -Dcom.ibm.ssl.keyStore=/home/user1/etc/key.p12
   -Dcom.ibm.ssl.trustStore=/home/user1/etc/trust.p12
   ```

   Recommended SSL configuration settings when running the application with a non-IBM product JRE:

   ```
   com.ibm.ssl.protocol=SSL
   com.ibm.ssl.trustManager=SunX509
   com.ibm.ssl.keyManager=SunX509
   com.ibm.ssl.contextProvider=SunJSSE

   com.ibm.ssl.keyStoreType=JKS
   com.ibm.ssl.keyStoreProvider=SUN
   com.ibm.ssl.keyStore=/home/user1/etc/key.jks

   com.ibm.ssl.trustStoreType=JKS
   com.ibm.ssl.trustStoreProvider=SUN
   com.ibm.ssl.trustStore=/home/user1/etc/trust.jks
   ```

   The key store file and trust store file must be created using the Java keytool utility before the application runs. The automatic key file generation is not supported with a non-IBM product JRE.

   You must override the default ORB implementation of the non-IBM product JRE with the `com.ibm.ws.orb_7.0.0.jar` file, or add it to the classpath.

What to do next

Enable trace for the IBM Thin Client for EJB by adding the following to the Java command.

```
-Dcom.ibm.ejs.ras.lite.traceSpecification=*=all
```

Related tasks

Example 1: Configuring basic authentication and identity assertion
This example presents a pure Java client, C, that accesses a secure enterprise bean on server, S1, through user bob. The following steps take you through the configuration of C, S1, and S2.

Using JMS to connect to a WebSphere Application Server default messaging provider messaging engine

Related reference

ssl.client.props client configuration file
Use the `ssl.client.props` file to configure Secure Sockets Layer (SSL) for clients. In previous releases of WebSphere Application Server, SSL properties were specified in the `sas.client.props` or `soap.client.props` files or as system properties. By consolidating the configurations, WebSphere Application Server enables you to manage security in a manner that is comparable to server-side configuration management. You can configure the `ssl.client.props` file with multiple SSL configurations.

Installing Application Client for WebSphere Application Server

This topic describes how to install the Application Client for WebSphere Application Server using the installation image on the product CD-ROM.

Before you begin

Running client applications that communicate with a WebSphere Application Server requires that elements of the Application Server are installed on the system on which the client applications run. However, if the system does not have the Application Server installed, you can install Application Client, which provides a stand-alone client run-time environment for your client applications.

The i5/OS Application Client supports the following types of client:

- Java thin client
- Java Platform, Enterprise Edition (Java EE) application client
- Web services thin client

The steps that follow provide enough detail to guide you through preparing for, choosing, and installing the variety of options and features provided. To prepare for installation and to make yourself familiar with installation options, complete the steps in this topic and read the related topics, before you start to use the installation tools. Specifically, read these topics before installing the product:

- Installing WebSphere Application Client from a Windows workstation
- Installing WebSphere Application Client from an i5/OS Qshell command line
- Best practices for installing

It is recommended that you have 350 MB of temporary disk space available before beginning the installation.

1. Install WebSphere Application Client from your iSeries server. Installing the product from the CD-ROM drive of your iSeries server requires direct physical access to the server. The local installation requires less time to complete than a remote installation. The local installation reads installation options from a response file.

   See Installing WebSphere Application Client from your iSeries server.
2. Run the GUI installation tool from a Windows workstation

   The interactive GUI uses an InstallShield interface to prompt you for installation options. Installing the product from a Windows workstation does not require direct physical access to the iSeries server.

   See “Installing Application Client for WebSphere Application Server from a Windows workstation” on page 245.

Results

You successfully installed the Application Client for WebSphere Application Server and the features you selected.

What to do next

Use the installation verification utility to verify a successful installation. If the installation is not successful, fix the error as indicated in the installation error message. For example, if you do not have enough disk space, add more space, and reinstall the Application Client.

Best practices for installing Application Client for WebSphere Application Server

This topic provides tips for installing Application Client on multiple platforms.

All platforms

The following tips pertain to installing Application Client on all platforms:

- Reserve at least 4 to 5MB free space in the target platform temporary directory.
- When specifying a different temporary directory while installing Application Client, the following message is displayed if the target platform default temporary directory does not have enough free space to install Application Client:

  ```
  Error writing file =  There may not be enough
  temporary disk space.
  Try using -is:tempdir to use a temporary
  directory on a partition with more disk
  space.
  ```

  Use the `-is:tempdir` installation option to specify a different temporary directory. For example, the following command uses `/swap` as a temporary directory during installation:

  ```
  ./install -is:tempdir /swap
  ```

- After the installation, when changing the installation settings for the WebSphere Application Server host name and the port number, edit the setupClient file.

  Change the `DEFAULTSERVERNAME` and `SERVERPORTNUMBER` to the new WebSphere Application Server host name and port number, respectively. If the `SERVERPORTNUMBER` is not set, then the default is 2809. Review the following example:

  ```
  set DEFAULTSERVERNAME=NDServerName
  set SERVERPORTNUMBER=9810
  ```

  The setupClient file is located in the `bin` subdirectory under the Application Client installation destination.

Specific platforms

The following tips pertain to installing Application Client on the designated platforms:

Installing Application Client for WebSphere Application Server silently

This topic provides the steps necessary to use the installation wizard and perform a silent installation.

About this task

Use these steps to perform a silent installation, which uses the installation wizard to install the product. Instead of displaying a user interface, the silent installation provides interaction between you and the wizard by reading all of your responses from a file that you must customize.

1. Verify that the host server jobs are started on your iSeries server. The host server jobs allow the installation code to run on iSeries. Enter this command on a CL command line: `STRHOSTSVR SERVER(*ALL)`
2. Verify that your user profile has the *ALLOBJ and *SECADM special authorities.
3. Place the WebSphere Application Server **Supplements** disc in the disc drive of your iSeries server.
4. Customize the options response file.
   a. Locate the sample options response file. The file name is setup.response in the operating system platform directory on the product CD-ROM.
   b. Use the Copy (CPY) command to create a copy of the setup.response file from the disk. For example:

      ```
      CPY OBJ('/QOPT/WEBSPHERE/AppClient/setup.response') TODIR('/MYDIR')
      ```

      Where:

      QOPT is the disk mount point.

      WEBSPHERE is the disk volume label.

      /AppClient is the product directory on the disk. This will be referred to in later steps.

      Note: For DVD media, an additional directory prefix is required before the product directory that reflects the operating system and architecture of the installation. Using the previous example, the following installation path applies when you use CD-ROM media:

      ```
      CPY OBJ('/QOPT/WEBSPHERE/AppClient/setup.response') TODIR('/MYDIR')
      ```

      However, the following installation path applies when you use DVD media:

      ```
      CPY OBJ('/QOPT/WEBSPHERE/os400_ppc64/AppClient/setup.response') TODIR('/MYDIR')
      ```

   c. Edit the copy of the file using EDTF from the i5/OS command line or using Wordpad from a mapped drive. Accept the license agreement by making the following change in the response file: `-OPT silentInstallLicenseAcceptance="true'` This is the only required change in the response file. Read the information in the response file to determine if you want to change any other options.
   d. Save the file.
5. Invoke the installation program for WebSphere Application Client for i5/OS. To do this, run the INSTALL command from Qshell :
   a. From the i5/OS command line, start a Qshell session: STRQSH
   b. Change directory to the AppClient subdirectory of CD-ROM or DVD-ROM drive: cd /QOPT/WEBSPHERE/AppClient **OR** cd /QOPT/WebSphere/os400_ppc64/AppClient
   c. Issue the INSTALL command to start the installation program: `install –options /MYDIR/setup.response`

      Important: Do not exit the Qshell session (PF3) until the installation has completed. Doing so might cause the installation to stop prematurely.

      A JDK is automatically selected for invoking the installer and configuring the product based on what JDKs are installed on the machine. The priority for selection is done in the following order:

      - IBM Technology for JDK 6 - 32 bit (5722-JV1 or 5761JV1 option 11)
      - IBM Technology for JDK 6 - 64 bit (5722-JV1 or 5761JV1 option 12)
      - IBM Developer Kit for Java 1.6 (5722-JV1 or 5761JV1 option 10)

      To force IBM Technology for JDK 6 - 64 bit add the parameter: `-devkit16_64`.

      To force IBM Developer Kit for Java 1.6 add the parameter: `-devkit16_CLASSIC`.

      Consider the following example to force the use of IBM Technology for JDK 6 - 64 bit:

      ```
      install –options /MYDIR/setup.response -devkit16_64
      ```

Results

You installed application clients silently by using the response file.

What to do next

Use the installation verification utility to verify a successful installation. If the installation is not successful, fix the error as indicated in the installation error message. For example, if you do not have enough disk space, add more space, and reinstall the Application Client.

Installing Application Client for WebSphere Application Server from a Windows workstation

This topic describes how to install the Application Client for WebSphere Application Server on your i5/OS system from a Windows workstation.

About this task

The steps that follow guide you through installing the WebSphere Application Client from a Windows workstation using the graphical installation wizard.

1. If TCP/IP is not started or if you don't know if TCP/IP is started, enter the Start TCP/IP (STRTCP) command on the Control Language (CL) command line.
2. Verify that the host server jobs are started on your iSeries server. The host server jobs allow the installation code to run on iSeries. Enter this command on a CL command line: STRHOSTSVR SERVER(*ALL)
3. Verify that your user profile has the *ALLOBJ and *SECADM special authorities.
4. Place the WebSphere Application Server **Supplements** disc in the disc drive of your workstation.
5. Install Application Client for WebSphere Application Server using the launchpad. Read the Using the launchpad to start the installation topic for more information.
6. Enter your user profile, password and the host name or IP address for the system you wish to install when prompted. Click **OK** to continue.
7. Click Next to continue when the Welcome panel is displayed. Click the radio button beside the ″I accept the terms in the license agreement″ message if you agree to the license agreement, and click **Next** to continue.
8. The installation wizard checks the system for prerequisites. Click **Next** if you see a success message on the wizard panel. If a warning message is displayed on the panel, click **Cancel** to exit the installation wizard and install the proper prerequisites to the system.

   If more than one Java SE Development Kit 6 (JDK 6) option is installed on the machine, then select one of the following:

   - IBM Technology for JDK 6 - 32 bit (5722-JV1 or 5761JV1 option 11)
   - IBM Technology for JDK 6 - 64 bit (5722-JV1 or 5761JV1 option 12)
   - IBM Development Kit for Java 1.6 (5722-JV1 or 5761JV1 option 10)

   Note: Application Client may be fully functional even if some prerequisites are not installed on the system, however it is recommended you update your system to the required level.
9. Specify the installation directory and the default profile directory. Click **Next** to continue.
   a. Specify the target directories for the Application Client product and default profile if you do not want to use the default directories.
   b. Deleting the default target location and leaving an installation directory field empty prevents you from continuing the installation process.

10. Choose a type of installation, and click **Next**. If you use the GUI you can choose a Typical installation type, which installs J2EE and Thin application client, or you can choose the custom installation type. For i5/OS, the custom installation type allows you to include the Web Services thin client and Administration thin client features.
    a. Install the Administration Thin Client, if you need a runtime jar that is customized to enable client applications to perform WebSphere administration tasks. The Administration Thin Client is installed into *app_server_root*/`runtimes`.
    b. Install the Web Services Thin Client, if you need a JAX-RPC Web services runtime JAR that is customized to enable client applications to communicate with the Application Server through Web Services. The Web Services Thin Client is installed into *app_server_root*/`runtimes`.
11. Enter the host name of the WebSphere Application Server machine. Click **Next** to continue. The default port number is 2809. If you do not know the host name of the system on which the application server will run, enter any string to proceed with the install.
12. Review the summary information, and click **Next** to install the product code or you might also click **Back** to change your specifications.
13. Click **Finish** to exit the wizard, after the Application Client installs.
14. Verify the success of the installer program by examining the Completion panel and the `app_client_root/logs/install/log.txt` file for installation status. The installer program records the following indicators of success in the logs:

    `INSTCONFSUCCESS` indicates that the installation is successful and that no further log analysis is required.

    `INSTCONFFAILED` indicates an installation failure that you cannot retry or recover from without reinstalling.

Results

You successfully installed the Application Client for WebSphere Application Server and the features you selected.

What to do next

Use the installation verification utility to verify a successful installation. If the installation is not successful, fix the error as indicated in the installation error message. For example, if you do not have enough disk space, add more space, and reinstall the Application Client.

Uninstalling Application Client for WebSphere Application Server

This task describes using the uninstall program to uninstall the Application Client for WebSphere Application Server.

Before you begin

If you want to uninstall IBM Application Client for WebSphere Application Server manually, read the Uninstalling the product on i5/OS topic.

Note: If you have a feature pack installed, uninstalling the Application Client for WebSphere Application Server using the Version 6.1.0 uninstaller program causes the feature pack to stop working because the uninstallation removes the server. However, you can uninstall the feature pack after uninstalling the WebSphere Application Server product.

Note: If you are logged in as a root user and attempt to uninstall an instance of Application Client for WebSphere Application Server that a non-root user installed, the Windows registry will also be

wiped clean for the application client instance that the root user installed. If you need to uninstall an instance of application client that a non-root user has installed, be sure that you are logged in as the non-root user.

1. Stop any browsers and any Java processes related to Application Client products.

   Read the Uninstalling the product on i5/OS topic for more information.
2. Change directories to the `uninstall` directory before issuing the command to uninstall the application client. The command file is located in the `uninstall` directory.
3. Issue the command to uninstall the product.

   Use the **uninstall all** command.

   The Uninstall wizard begins and displays the Welcome panel.
4. Click **Next** to begin uninstalling the product. The Uninstall wizard displays a Confirmation panel that lists the product and features that you are uninstalling.
5. Click **Next** to continue uninstalling the product. The Uninstall wizard deletes existing profiles first.

   After deleting profiles, the Uninstall wizard deletes core product files by component.
6. Click **Finish** to close the wizard after the wizard removes the product.

Results

Application Client for WebSphere Application Server is uninstalled.

What to do next

Verify the uninstall procedure by viewing the *app_server_root*`/logs/uninstall/log.txt` file for errors. Look for the `INSTCONFSUCCESS`, indicating a successful uninstall in the log file:

```
Uninstall, com.ibm.ws.install.ni.ismp.actions.ISMPLogSuccessMessageAction, msg1,
   INSTCONFSUCCESS
```

The uninstallation wizard records uninstallation events in the uninstallation log files. Examine messages that the uninstallation program displays. If the Application Client for WebSphere Application Server does not uninstall successfully, read the messages to identify why the uninstallation failed. Correct the problems identified and try uninstalling the product again.

Note: If you are logged in as non-root or non-admin user, the uninstall may not be successful. When the uninstaller program runs, a log file is created in your home directory. If the uninstaller program fails, an attempt is made to move the log to the *app_server_root* directory; however, without the necessary permission to move the file to the *app_server_root* directory, the attempt to move fails and the log remains in your home directory. Look for the following log files in the *user_home*`/waslogs` directory:

- `log.txt`
- `trace.txt.gz`
- `trace.xml.gz`

Uninstalling IBM Application Client for WebSphere Application server on i5/OS

You can uninstall IBM Application Client for WebSphere Application Server for i5/OS product by running commands from your iSeries server.

About this task

Uninstall IBM Application Client for WebSphere Application Server for i5/OS from your iSeries server. To uninstall the product, run the following command from the Qshell command line:

app_client_root`/bin/uninstall`

Uninstalling IBM Application Client for WebSphere Application Server from the iSeries server removes all the product libraries and directories. If you later decide that you do not need the user-defined data, you can manually remove it.

Results

After completing the procedure, the IBM Application Client for WebSphere Application Server product is uninstalled.

installRegistryUtils command

The installRegistryUtils command-line tool runs on any supported operating system. The installRegistryUtils tool lists installed products and packages and assists in cleaning a particular registered product (including the associated packages), package, or all registered products (including all associated packages) which are visible to the current user.

Location of the command file: The command file is located on both the product disc and the installed product.

Note: When using the product disc, the command must be invoked on the machine targeted by the command. The tool does not work remotely.

To invoke the command file from the product disc, run the following command file from the root of the product CD:

- `installRegistryUtils/bin/installRegistryUtils`

To invoke the command file from the installed product, use the following command:

- *app_server_root*`/bin/installRegistryUtils`

The common location for the command file is /QIBM/WAS/bin/installRegistryUtils.

Syntax for the installRegistryUtils command: Clean up all accessible products and packages.

```
./installRegistryUtils -cleanAll
 [-userHome user_home]
```

Clean up the specified product and all associated packages.

```
./installRegistryUtils -cleanProduct
  -offering offering_ID
  -installLocation installation_location
 [-userHome user_home]
```

Clean up the specified package.

```
./installRegistryUtils -cleanPackage
  -pakName package_name
  -installLocation installation_location
 [-userHome user_home]
```

List all visible products.

```
./installRegistryUtils -listProducts
 [-userHome user_home]
```

List all visible packages.

```
./installRegistryUtils -listPackages
 [-userHome user_home]
```

Parameters: Supported arguments include:

-installLocation *installation_location*
Specifies the full path to the product installation.

-offering *offering_ID*
Identifies the offering ID, also known as the product ID, of the specified product.

-pakName *package_name*
Identifies a package name of the specified product.

-userHome *user_home*
Specifies the full path to a user home directory. This parameter is not required for a root user unless you want to invoke the installRegistryUtils command on the home directory of a non-root user.

Deploying J2EE application clients on workstation platforms

You can deploy the J2EE application clients on workstation platforms using the methods described in this topic.

Before you begin

After developing an application client, deploy this application on client machines. *Deployment* consists of pulling together the various artifacts that the application client requires.

The *Application Client Resource Configuration Tool* (ACRCT) defines resources for the application client. These configurations are stored in the client `.jar` file within the application `.ear` file. The application client run time uses these configurations for resolving and creating an instance of the resources for the application client.

Note: This task only applies to J2EE application clients. Only perform this task if you configured your J2EE application client to use resource references.

About this task

1. Start the ACRCT and open an EAR file.
2. Configure new data source providers.
3. Configure mail providers and sessions.
4. Configure URL providers and sessions.
5. Configure Java messaging resources.
6. Configure new environment entries.
7. (Optional) Remove application client resources.
8. Save the EAR file.

Resource Adapters for the client

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application client and provides connectivity between the EIS and the enterprise application.

The resource adapter support for the J2EE client applications is a subset of the support for the server. For any resource adapter installed using the clientRAR tool, the client resource adapter is used in a non-managed environment and must conform to the J2EE Connector Architecture Specification Version 1.5 or higher. Only outbound connections to the EIS are supported through the ManagedConnectionFactory interfaces. The inbound messaging support (from the EIS), life cycle management, and work management aspects of the specification are not supported on the client.

For a client application to use a resource adapter, it must be installed in the directory specified by the environment variable, CLIENT_CONNECTOR_INSTALL_ROOT, defined when the setupCmdLine script

runs. The launchClient tool, Application Client Resource Configuration Tool (ACRCT) and clientRAR tool all use this variable to find the default location of all installed resource adapters. To install a resource adapter in the client, use the clientRAR tool. Once the resource adapter is installed, it must be configured using the ACRCT. The client configuration tool adds the resource adapter configuration to the EAR file. Then, connection factories and administered objects are defined.

When running J2EE application clients, the `launchClient` script specifies a system property called `com.ibm.ws.client.installedConnector`, which is set to the same value as the `CLIENT_CONNECTOR_INSTALL_ROOT` variable. This is the default location for installed resource adapters and can be overridden for each `launchClient` call by specifying the `-CCD` parameter. When the client container is activated, all resource adapter subdirectories under the specified default location for the resource adapters directory are added to the classpath. This action allows the client application to use the resource adapters without using the ACRCT to specify any of the client resources.

Using resource adapters is a new mechanism for easily extending client applications.

Configuring resource adapters

Use the Application Client Resource Configuration Tool (ACRCT) to configure resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure new resource adapters. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new resource adapters from the tree.
4. Expand the JAR file to view its contents.
5. Right-click the **Resource Adapters** folder, and click **New**.
6. Configure the resource adapter settings in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save** on the menu bar to save your changes.

clientRAR tool

This topic describes the command line syntax for the client resource adapter installation tool.

If this tool is used to add or delete resource adapters on the server, then only the client can use the resource adapter. If the resource adapter is installed on the server using the wsadmin tool or the administrative console, then do not use the clientRAR tool remove it. Only resource adapters that are installed using the clientRAR tool should be removed using the clientRAR tool.

The command line invocation syntax for the `clientRAR` tool follows:

```
clientRAR [-help | -?] [-CRDcom.ibm.ws.client.installedConnectors=<dir>] <task> <archive>

where
-help, -?
Print the usage information.
-CRDcom.ibm.ws.client.installedConnectors
The directory where resource adapters are installed.
This will override the system property of the same name
(com.ibm.ws.client.installedConnectors).


<task>
The task to perform: add - install, delete - uninstall.

<archive>
if task=add then this is the fully qualified name of the resource adapter archive file.
If task=delete then this is the filename of the resource adapter archive to be uninstalled.
```

The following examples demonstrate correct syntax.

Configuring new connection factories for resource adapters

Use the Application Client Resource Configuration Tool (ACRCT) to configure new connection factories for resource adapters.

About this task

Complete this task to configure new connection factories for resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure new connection factories. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new connection factories from the tree.
4. Expand the JAR file to view its contents.
5. Click the **Resource Adapters** folder.
6. Expand the resource adapter for which you want to create connection factories.
7. Right-click the **Connection Factories** folder and click **New**.
8. Configure the connection factory properties in the resulting property dialog.
9. Click **OK**.
10. Click **File** > **Save** on the menu bar to save your changes.

Resource adapter connection factory settings:

Use this panel to view or change the configuration properties of the selected resource adapter connection factory.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapters**. Right-click the **Connection Factories** folder, and click **New**. The following fields appear on the **General** tab.

Name:

The name by which this connection factory is known for administrative purposes within WebSphere Application Server. The name must be unique within the resource adapter connection factories across the product administrative domain.

Data type String

Description:

An optional description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this resource adapter connection factory definition to the deployment descriptor. This entry should be a `resource-ref` name.

Data type String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly when getting a connection. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly when getting a connection.

Data type String

Password:

Specifies an encrypted password. If you complete this field, then the **Password** field in the Properties box is ignored.

If you specify a value for the **UserName** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Type:

A drop-down list of all the `connectionFactoryInterfaces` as defined for the factories in the Resource Adapter Archive.

For each **Type**, there is a set of properties specified in the Properties box. This set of properties is constructed by retrieving the properties from each connection definition object. For any existing connection factories that are displayed for updating, this list of properties is overlaid with the properties specified for the objects. When the **Type** field is changed, the properties also change to reflect the correct properties for that type.

Data type String

Configuring administered objects

This section helps you configure new administered objects.

Before you begin

Before you configure new administered objects, you must complete the following prerequisites:

1. Install the Resource Adapter Archive file (RAR) using the clientRAR tool.
2. Configure the resource adapter for the `.ear` file, using the Application Client Resource Configuration Tool (ACRCT) tool.

About this task

Complete this task to configure new administered objects for installed resource adapters.

1. Start the Application Client Resource Configuration Tool (ACRCT).

2. Open the EAR file for which you want to configure new administered objects. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new administered objects from the tree.
4. Expand the JAR file to view its contents.
5. Click the **Resource Adapters** folder.
6. Expand the resource adapter for which you want to create administered objects.
7. Right-click the **Administered Objects** folder and click **New**.
8. Configure the administered object properties in the resulting property dialog.
9. Click **OK**.
10. Click **File** > **Save** on the menu bar to save your changes.

Administered objects settings:

Use this panel to view or change the configuration properties of the selected administered objects.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapters** > *resource_adapter_instance*. Right-click **Administered Objects** and click **New**. The following fields appear on the **General** tab.

The settings for administered objects are handled similarly to connection factories. When updating administered objects, use the same panels that you used to create administered objects.

Name:

The name by which this administered object is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the resource adapter administered objects across the product administrative domain.

Data type String

Description:

An optional description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

This entry is a `resource-env-ref` name, a `message-destination-ref` name (if the `message-destination-ref` has no link), or a `message-destination` link.

Data type String

Type:

A drop-down list of all the administered object class-interface pairs as defined for the admin objects in the Resource Adapter Archive (RAR) file.

For each **Type**, there is a set of properties specified in the Properties box. This set of properties is constructed by retrieving the properties from each administered object definition. For any existing administered objects that are displayed for updating, this list of properties is overlaid with the properties

specified for the objects. When the **Type** field is changed, the properties also change to reflect the correct properties for that type.

Data type String

Resource adapter settings

Use this panel to view or change the configuration properties of the resource adapter. These configuration properties control how resource adapters are created.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Adapter**. Right-click **Resource Adapter** and click **New**. The following fields appear on the **General** tab.

Name

The name by which this Resource Adapter is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the Resource Adapters across the product administrative domain.

Data type String

Description

A description of this resource adapter for administrative purposes within IBM WebSphere Application Server.

Data type String

Class Path

Any additional class path. The path to the resource adapter directory is automatically added.

Data type String
Default The path to your Resource Adapter directory.

Native Path

The native path where the Resource Adapter is located. Enter any additional native class path here.

Data type String

Resource Adapter Name

A mandatory field that points to an installed resource adapter subdirectory. The entry does not represent the full directory name for the resource adapter. The full directory name is the installed resource adapter path, plus the resource adapter name.

Data type String

Installed Resource Adapter Path

The directory where resource adapters are installed. If you do not complete this field, then the default takes effect.

If you specify the value, `${CONNECTOR_INSTALL_ROOT}`, then this value replaces the value of the `CLIENT_CONNECTOR_INSTALL_ROOT` variable on the machine on which the client application runs. This action allows the application to run easily on different machines, where the client installation might be in different locations.

Data type String
Default ${CONNECTOR_INSTALL_ROOT}

Starting the Application Client Resource Configuration Tool and opening an EAR file

You can perform many tasks by starting the Application Client Resource Configuration Tool (ACRCT). Many of these tasks also involve then opening an EAR file.

Before you begin

Note: This task only applies to J2EE application clients.

About this task

Use these steps to start the Application Client Resource Configuration Tool. When you start the tool, one of the most common tasks that you perform is opening and modifying the components of EAR files.

1. Open a command prompt and change to the *app_server_root*\bin directory.
2. Run the clientConfig.bat file.
3. Open an EAR file within the Application Client Resource Configuration Tool (ACRCT):
   a. Click **File** > **Open**.
   b. Select the file and click **Open**.
   •
4. Save your changes to the file and close the tool:
   a. Click **File** > **Save**.
   b. Click **File** > **Exit**.
   •

Data sources for the Application Client

WebSphere Application Server and the Application Client for WebSphere Application Server do not provide client database drivers to be used directly from a J2EE application client. If your application client accesses a database directly, you must provide the database drivers on the client machine.

You can contact your database vendor to acquire client database driver code and licenses. In addition, data sources configured on the server and looked up on the client do not participate in global transactions. Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine because the database access is handled by the enterprise bean running on WebSphere Application Server. For a current list of providers that are supported on WebSphere Application Server visit the WebSphere Application Server prerequisite Web site. (See the following link.)

Data source properties for application clients

Use this page to create or modify the data sources.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Data Source Providers** > *Data source provider instance*. Right-click **Data Sources** and click **New**. The following fields are displayed on the **General** tab:

Name

Specifies the display name of this data source.

Data type String

Description
Specifies a text description of the data source.

Data type String

JNDI Name
The application client run time uses this field to retrieve configuration information.

Database Name
The name of the database to which you want to connect.

User
Use the user ID with the Password property, for authentication if the calling application does not provide a user ID and password explicitly.

If you specify a value for the User ID property, then you must also specify a value for the Password property. The connection factory User ID and Password properties are used if the calling application does not provide a user ID and password explicitly.

Password
Use the password with the User ID property, for authentication if the calling application does not provide a user ID and password explicitly.

If you specify a value for the Password property, then you must also specify a value for the User ID property.

Re-Enter Password
Confirms the password.

Custom Properties
Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new data source providers (JDBC providers) for application clients
You can create new data source providers, also known as JDBC providers, for your application client using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

During this task, you create new data source providers, also known as JDBC providers, for your application client. In a separate administrative task, install the Java code for the required data source provider on the client machine on which the application client resides.

About this task

Use this task to connect application clients to relational databases.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file for which you want to configure the new data source provider. The EAR file contents display in a tree view.
2. Select the JAR file in which you want to configure the new data source provider from the tree.

3. Expand the JAR file to view its contents.
4. Click the **Data Source Providers** folder. Do one of the following:
   - Right-click the folder and click **New Provider**.
   - Click **Edit > New** on the menu bar.
5. Configure the data source provider properties in the resulting property dialog.
6. Click **OK** when you finish.
7. Click **File > Save** on the menu bar to save your changes.

Example: Configuring data source provider and data source settings

You can configure data source provider and data source settings.

The purpose of this article is to help you to configure data source provider and data source settings.

- Required fields:
  - Data Source Provider Properties page: name
  - Data Source Properties page: name, jndiName
- Special cases:
  - The user name and password fields have no equivalent XMI tags. You must specify these fields in the custom properties.
  - The password is encrypted when you use the Application Client Resource Configuration Tool (ACRCT). If you do not use the ACRCT the field cannot be encrypted.
- Example:

```
<resources.jdbc:JDBCProvider xmi:id="JDBCProvider_1" name="jdbcProvider:name"
description="jdbcProvider:description" implementationClassName="jdbcProvider:
ImplementationClass">
<classpath>jdbcProvider:classPath</classpath>
<factories xmi:type="resources.jdbc:WAS40DataSource" xmi:id="WAS40DataSource_1"
name="jdbcFactory:name" jndiName="jdbcFactory:jndiName"
description="jdbcFactory:description" databaseName="jdbcFactory:databasename">
<propertySet xmi:id="J2EEResourcePropertySet_13">
<resourceProperties xmi:id="J2EEResourceProperty_13" name="jdbcFactory:customName"
value="jdbcFactory:customValue"/>
<resourceProperties xmi:id="J2EEResourceProperty_14" name="user"
value="jdbcFactory:user"/>
<resourceProperties xmi:id="J2EEResourceProperty_15" name="password"
value="{xor}NTs9PBk+PCswLSZlMT4yOg==/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_14">
<resourceProperties xmi:id="J2EEResourceProperty_16" name="jdbcProvider:customName"
value="jdbcProvider:customeValue"/>
</propertySet>
</resources.jdbc:JDBCProvider>
```

Data source provider settings for application clients

Use this page to create a data source under a JDBC provider which provides the specific JDBC driver implementation class.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Data Source Providers** > and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the display name for the data source.

For example you can set this field to *Test Data Source*.

Data type String

Description:

Specifies a text description for the resource.

Data type	String

Class Path:

A list of paths or `.jar` file names which together form the location for the resource provider classes.

Implementation class:

Use this setting to perform database specific functions.

Data type	String
Default	Dependent on JDBC driver implementation class

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new data sources for application clients

About this task

During this task, you create new data sources for your application client.

1. Click the data source provider for which you want to create a data source in the tree. Take one of the following actions as needed:
   - Configure a new data source provider.
   - Click an existing data source provider.
2. Expand the data source provider to view its **Data Sources** folder.
3. Click the data source folder. Take one of the following actions as needed:
   - Right click the data source folder and click **New Factory**.
   - Click **Edit > New** on the menu bar.
4. Configure the data source properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring mail providers and sessions for application clients

You can edit the configurations of mail sessions and providers for your application clients using the Application Client Resource Configuration Tool (ACRCT).

About this task

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of mail sessions and providers for your application clients to use.

1. Start the ACRCT.
2. Open an EAR file.

3. Locate the mail objects in the tree that is displayed for the EAR file. For example, if your file contains mail sessions, expand **Resources** → ***application*.jar** → **Mail Providers** → ***java_mail_provider_instance*** → **Mail Sessions**.

   In this example, ***java_mail_provider_instance*** is a particular mail provider.

Results

The mail session instances are located in the **JavaMail Sessions** folder.

Mail provider settings for application clients

Use this page to implement the JavaMail API and create mail sessions.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right-click **Mail Providers** > and click **New**. The following fields appear on the **General** tab:

Name:

The name of the JavaMail resource provider.

Description:

An optional description for the resource provider.

Class Path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Specifies the name of the protocol.

Classname:

Specifies the name of the class implementing the protocol. Leave this field blank if you want to use the default implementation.

Type:

This menu contains the following two values: `TRANSPORT` or `STORE`.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Mail session settings for application clients

Use this page to configure mail session properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Mail Providers** > *mail provider instance*. Right-click **Mail Sessions** and click **New**. The following fields appear on the **General** tab:

Name:

Represents the administrative name of the JavaMail session object.

Description:

Provides an optional description for your administrative records.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Mail Transport Host:

Specifies the server to connect to when sending mail.

Mail Transport Protocol:

Specifies the transport protocol to use when sending mail.

Mail Transport User:

Specifies the user ID to use when the mail transport host requires authentication.

Mail Transport Password:

Specifies the password to use when the mail transport host requires authentication.

Enable strict Internet address parsing:

Specifies whether the recipient addresses must be parsed strictly in compliance with RFC 822, which is a specifications document issued by the Internet Architecture Board.

This setting is not generally used for most mail applications. RFC 822 syntax for parsing addresses effectively enforces a strict definition of a valid e-mail address. If you select this setting, JavaMail will adhere to RFC 822 syntax and reject recipient addresses that do not parse into valid e-mail addresses (as defined by the specification). If you do not select this setting, JavaMail will not adhere to RFC 822 syntax and will accept recipient addresses that do not comply with the specification. By default, this setting is deselected. You can view the RFC 822 specification at the following URL for the World Wide Web Consortium (W3C): http://www.w3.org/Protocols/rfc822/.

Re-Enter Password:

Confirms the password.

Mail From:

Specifies the mail originator.

Mail Store Host:

Specifies the mail account host (or ″domain″) name.

Mail Store User:

Specifies the user ID of the mail account.

Mail Store Password:

Specifies the password of the mail account.

Re-Enter Password:

Confirms the password.

Mail Store Protocol:

Specifies the protocol to be used when receiving mail.

Mail Debug:

When `true`, JavaMail interaction with mail servers, along with these mail session properties are printed to the `stdout` file.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring mail provider and mail session settings for application clients

You can configure mail provider and mail session settings. This topic provides the required fields, special cases, and an example.

The purpose of this topic is to help you configure mail provider and mail session settings.

- Required fields:
  - Mail Provider Properties page: name, and at least one protocol provider
  - Mail Session Properties page: name, jndiName, outgoing server and protocol, and/or incoming server and protocol
- Special cases:
  - If you use the ACRCT tool, the password field will be encrypted. You cannot encrypt the password field if you do not use the ACRCT tool.
- Example:

```
<resources.mail:MailProvider xmi:id="builtin_mailprovider" name="Built-in Mail Provider" description="The built-in mail provider">
  <factories xmi:type="resources.mail:MailSession"
             xmi:id="MailSession_1207766754834" name="MailSession"
             jndiName="mail/session" description="Sample mail session" category="Sample"
             mailTransportHost="smtp.coldmail.com" mailTransportUser="transportUser"
             mailTransportPassword="{xor}Lz4sLChvLTs="
             mailFrom="smith@coldmail.com" mailStoreHost="imap.coldmail.com" mailStoreUser="storeUser"
             mailStorePassword="{xor}Lz4sLChvLTs="
             debug="true" strict="true"
             mailTransportProtocol="builtin_smtp" mailStoreProtocol="builtin_imap">
    <propertySet xmi:id="J2EEResourcePropertySet_1207766778585">
      <resourceProperties xmi:id="J2EEResourceProperty_1207766778585" name="key" type="java.lang.String" value="value" required="false"/>
    </propertySet>
  </factories>
  <protocolProviders xmi:id="builtin_smtp" protocol="smtp" classname="com.sun.mail.smtp.SMTPTransport" type="TRANSPORT"/>
  <protocolProviders xmi:id="builtin_pop3" protocol="pop3" classname="com.sun.mail.pop3.POP3Store" type="STORE"/>
  <protocolProviders xmi:id="builtin_imap" protocol="imap" classname="com.sun.mail.imap.IMAPStore" type="STORE"/>
```

```
  <protocolProviders xmi:id="builtin_smtps" protocol="smtps" classname="com.sun.mail.smtp.SMTPSSLTransport" type="TRANSPORT"/>
  <protocolProviders xmi:id="builtin_pop3s" protocol="pop3s" classname="com.sun.mail.pop3.POP3SSLStore" type="STORE"/>
  <protocolProviders xmi:id="builtin_imaps" protocol="imaps" classname="com.sun.mail.imap.IMAPSSLStore" type="STORE"/>
</resources.mail:MailProvider>
```

Configuring new mail sessions for application clients

You can use the Application Client Resource Configuration Tool (ACRCT) to configure new mail sessions for your application client.

Before you begin

During this task, you configure new mail sessions for your application client. The mail sessions are associated with the pre-configured default mail provider supplied by the product.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the EAR file. The EAR file contents are displayed in a tree view.
2. Select the JAR file in which you want to configure the new JavaMail session.
3. Expand the JAR file to view its contents.
4. Click **Mail Providers** > **Mail Provider** > **Mail Sessions**. Complete one of the following actions:
   - Right click the **Mail Sessions** folder and select **New Factory**.
   - Click **Edit** > **New** on the menu bar.
5. Configure the Mail Session properties in the displayed fields.
6. Click **OK**.
7. Click **File** > **Save** on the menu bar to save your changes.

URLs for application clients

A *Uniform Resource Locator* (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as `http`, `ftp`, `file`, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The scheme_information for HTTP, FTP and File generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

```
http://www.ibm.com/software/webservers/appserv/library.html.
```

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL providers for the Application Client Resource Configuration Tool

A URL provider implements the function for a particular URL protocol, such as HyperText Transfer Protocol (HTTP). This provider, comprised of a pair of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Configuring new URL providers for application clients

You can create URL providers and URLs for your client application using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

During this task, you create URL providers and URLs for your client application. In a separate administrative task, you must install the Java code for the required URL provider on the client machine on which the client application resides.

About this task

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new URL provider. The EAR file contents display in a tree view.
3. Select the JAR file in which you want to configure the new URL provider from the tree.
4. Expand the JAR file to view the contents.
5. Click the folder called **URL Providers**. Complete one of the following actions:
   - Right click the folder and select **New**.
   - Click **Edit** > **New** on the menu bar.
6. Configure the URL provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save** on the menu bar to save your changes.

Configuring URL providers and sessions using the Application Client Resource Configuration Tool

You can edit the configurations of URL providers and URLs to be used by your application clients using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

Use the Application Client Resource Configuration Tool (ACRCT) to edit the configurations of URL providers and URLs to be used by your application clients.

About this task

1. Start the ACRCT.
2. Open an EAR file.
3. Locate the URL objects in the tree that displays. For example, if your file contains URL providers and URLs, expand **Resources** -> ***application*.jar** -> **URL Providers** -> ***url_provider_instance***

   where ***url_provider_instance*** is a particular URL provider.
4. If you expand the tree further, you will also see the **URLs** folders containing the URL instances for each URL provider instance.

URL settings for application clients:

Use this page to implement the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP).

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **URL Providers** > *URL provider instance*. Right-click **URLs** and click **New**. The following fields appear on the **General** tab.

This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

The administrative name for the URL.

Description:

This is an optional description of the URL for your administrative records.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

URL:

A Uniform Resource Locator (URL) name that points to an Internet or intranet resource. For example: `http://www.ibm.com`.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

URL provider settings for application clients:

Use this page create new URL providers.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right click **URL Providers**, and click **New**. The following fields appear on the **General** tab.

A URL provider implements the function for a particular URL protocol, such as Hyper Text Transfer Protocol (HTTP). This provider, comprised of classes, extends the `java.net.URLStreamHandler` and `java.net.URLConnection` classes.

Name:

Administrative name for the URL.

Description:

Optional description of the URL, for your administrative records.

Class Path:

A list of paths or JAR file names which together form the location for the resource provider classes.

Protocol:

Protocol supported by this stream handler. For example, `nntp`, `smtp`, `ftp`, and so on.

To use the default protocol, leave this field blank.

Stream handler class:

Fully qualified name of a User-defined Java class that extends the `java.net.URLStreamHandler` for a particular URL protocol, such as FTP.

To use the default stream handler, leave this field blank.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring URL and URL provider settings for application clients

You can configure URL and URL provider settings. This topic provides the required fields and an example.

The purpose of this article is to help you to configure URL and URL provider settings.

- Required fields:
  - URL Properties page: name, jndiName, url
  - URL Provider Properties page: name
- Example:

```
<resources.url:URLProvider xmi:id="URLProvider_1" name="urlProvider:name"
description="urlProvider:description"
streamHandlerClassName="urlProvider:streamHandlerClass"
protocol="urlProvider:protocol">
<classpath>urlProvider:classpath</classpath>
<factories xmi:type="resources.url:URL" xmi:id="URL_1" name="urlFactory:name"
jndiName="urlFactory:jndiName" description="urlFactory:description"
spec="urlFactory:url">
<propertySet xmi:id="J2EEResourcePropertySet_18">
<resourceProperties xmi:id="J2EEResourceProperty_20" name="urlFactory:customName"
value="urlFactory:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_19">
<resourceProperties xmi:id="J2EEResourceProperty_21" name="urlProvider:customName"
value="urlProvider:customValue"/>
</propertySet>
</resources.url:URLProvider>
```

Configuring new URLs with the Application Client Resource Configuration Tool

You can use URLs for your client application using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

During this task, you create URLs for your client application.

About this task

1. Click the URL provider for which you want to create a URL in the tree. Complete one of the following:
   - Configure a new URL provider.
   - Click an existing URL provider.

2. Expand the URL provider to view the **URLs** folder.
3. Click the URL folder. Complete one of the following actions:
   - Right click the folder and click **New**.
   - Click **Edit -> New** on the menu bar.
4. Configure the URL properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File** > **Save** in the menu bar to save your changes.

Asynchronous messaging in WebSphere Application Server using JMS

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interface. The JMS interface provides a common way for Java programs (clients and Java Platform, Enterprise Edition (Java EE) applications) to create, send, receive, and read asynchronous requests as JMS messages.

This topic provides a generic overview of asynchronous messaging using the JMS support provided by WebSphere Application Server.

The base support for asynchronous messaging using the JMS API provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This support enables WebSphere product Java EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients, by using JMS destinations (queues or topics). A Java EE application can use JMS queue destinations for point-to-point messaging and JMS topic destinations for publish and subscribe messaging. A Java EE application can explicitly poll for messages on a destination, and then retrieve messages for processing by business logic beans (enterprise beans).

With the base JMS and XA support, the Java EE application uses standard JMS calls to process messages, including any responses or outbound messaging. An enterprise bean can handle responses acting as a sender bean, or within the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction. This level of function for asynchronous messaging is called *bean-managed messaging*, and gives an enterprise bean complete control over the messaging infrastructure, for example, connection and session pool management. The common container has no role in bean-managed messaging.

WebSphere Application Server also supports automatic asynchronous messaging using message-driven beans (a type of enterprise bean defined in the Enterprise JavaBeans (EJB) 2.0 specification) and JMS listeners (part of the JMS application server facilities). Messages are automatically retrieved from JMS destinations, optionally within a transaction, then sent to the message-driven bean in a Java EE application, without the application having to explicitly poll JMS destinations.

Java Message Service providers for clients

This topic describes the different ways client applications can use Java Message Service (JMS) providers with WebSphere Application Server.

IBM WebSphere Application Server supports asynchronous messaging through the use of a JMS provider and its related messaging system. JMS providers must conform to the JMS specification version 1.1. To use message-driven beans the JMS provider must support the optional Application Server Facility (ASF) function defined within that specification, or support an inbound resource adapter as defined in the JCA specification version 1.5.

The service integration technologies of IBM WebSphere Application Server can act as a messaging system when you have configured a service integration bus that is accessed through the default messaging provider. This support is installed as part of WebSphere Application Server, administered through the administrative console, and is fully integrated with the WebSphere Application Server runtime.

WebSphere Application Server also includes support for the following JMS providers:

WebSphere MQ
Provided for use with supported versions of WebSphere MQ.

Generic
Provided for use with any 3rd party messaging system which supports ASF.

For backwards compatibility with earlier releases, WebSphere Application Server also includes support for the V5 default messaging provider which enables you to configure resources for use with the WebSphere Application Server version 5 Embedded Messaging system. The V5 default messaging provider can also be used with a service integration bus.

WebSphere applications can use messaging resources provided by any of these JMS providers. However the choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you may already have a messaging infrastructure based on WebSphere MQ. In this case you may either connect directly using the included support for WebSphere MQ as a JMS provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

The service integration bus also provides access to a default messaging provider. This is a J2EE 1.4 compliant JMS messaging provider which is fully integrated with WebSphere Application Server. You can use it in multiple server configurations for messaging interactions with a WebSphere MQ network.

Configuring Java messaging client resources

To configure Java messaging client resources, you create new JMS provider configurations for your application client. The application client can use a messaging service through the Java Message Service APIs. A JMS provider provides two kinds of J2EE factories. One is a *JMS connection factory*, and the other is a *JMS destination factory*.

Before you begin

In a separate administrative task, install the Java Message Service (JMS) client on the client machine where the application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

Note: When completing this task, you can either create a new messaging provider, or you can use an existing one.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new JMS provider. The EAR file contents are in the displayed tree view.
3. Select the JAR file in which you want to configure the new JMS provider from the tree.
4. Expand the JAR file to view its contents.
5. Optionally right-click **Messaging Providers** and select **New**, if you want to create and use a new messaging provider.
6. Configure the JMS provider properties in the resulting property dialog.
7. Click **OK**.
8. Click **File** > **Save**.

Configuring new JMS providers with the Application Client Resource Configuration Tool

You can create new Java Message Service (JMS) provider configurations for the Application Client. The Application Client makes use of a messaging service through the JMS interfaces.

Before you begin

During this task, you create new Java Message Service (JMS) provider configurations for the Application Client. The Application Client makes use of a messaging service through the JMS interfaces. A JMS provider provides two kinds of J2EE resources. One is a JMS connection factory, and the other is a JMS destination.

In a separate administrative task, you must install the JMS client on the client machine where your particular application client resides. The messaging product vendor must provide an implementation of the JMS client. For more information, see your messaging product documentation.

About this task

1. Start the Application Client Resource Configuration Tool and open the EAR file for which you want to configure the new JMS provider. The EAR file contents are displayed in a tree view.
2. From the tree, select the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Right-click **Messaging Providers**. Complete one of the following actions:
   - Right click the folder and select **New**.
   - On the menu bar, click **Edit > New**.
5. In the resulting property dialog, configure the JMS provider properties.
6. Click **OK** when finished.
7. Click **File -> Save** on the menu bar to save your changes.

JMS provider settings for application clients

Use this page to configure properties of the Java Message Service (JMS) provider, if you want to use a JMS provider other than the default messaging provider or the WebSphere MQ as a JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file. Right click **Messaging Providers**, and click **New**. The following fields appear on the **General** tab.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Description:

A description of the JMS provider, for administrative purposes.

Data type String

Class Path:

A list of paths or `.jar` file names which together form the location for the resource provider classes.

Context factory class:

The Java class name of the initial context factory for the JMS provider.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type String

Provider URL:

The JMS provider URL for external JNDI lookups.

For example, an LDAP URL for a JMS provider has the form: `ldap://hostname.company.com/contextName`.

Data type String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Default Provider connection factory settings

Use this panel to view or change the configuration properties of the selected JMS connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display the appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the connection factory.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this Resource Adapter connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type String

User Name:

The **User Name** used with the **Password** property for connecting to an application.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to authenticate connection to an application.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the connection factory connects.

Data type String

Client Identifier:

The name of the client. Required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Durable Subscription Home:

The name of the durable subscription home.

Data type String

Share durable subscriptions:

Controls whether or not durable subscriptions are shared across connections with members of a server cluster.

Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers.

Data type Selection list
Default In cluster
Range

In cluster
Allows sharing of durable subscriptions when connections are made from within a server cluster.

Always shared
Durable subscriptions can be shared across connections.

Never shared
Durable subscriptions are never shared across connections.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default Default
Range Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default BusMember
Range BusMember, Custom, ME

Target Significance:

The priority of significance for the target specified.

Default Preferred
Range Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example `merlin:7276:BootstrapBasicMessaging,Gandalf:5557:BootstrapSecureMessaging`

where

`BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).

Default

- If the host name is not specified, then the default `localhost` is used as a default value.
- If the port number is not specified, then `7276` is used as a default value.
- If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default	Bus
Range	Bus, Host, Cluster, Server

Temporary Queue Name Prefix:

The prefix to apply to the names of temporary queues. This name is a maximum of 12 characters.

Data type	String

Temporary Topic Name Prefix:

The prefix to apply to the names of temporary topics. This name is a maximum of 12 characters.

Data type	String

Default Provider queue connection factory settings

Use this panel to view or change the configuration properties of the selected JMS queue connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display the appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the queue connection factory.

Data type	String

Description:

A description of this queue connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String

JNDI Name:

The JNDI name that is used to match this queue connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type	String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to create an encrypted. If you complete this field, then the Password field in the Properties box is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the queue connection factory connects.

Data type String

Client Identifier:

The client identifier. Required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default Default
Range Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default BusMember
Range BusMember, Custom, Destination, ME

Target Significance:

The priority of significance for the target specified.

Default Preferred
Range Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example `localhost:7777:BootstrapBasicMessaging`

where

`BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).

Default

- If the host name is not specified, then the default `localhost` is used as a default value.
- If the port number is not specified, then `7276` is used as a default value.
- If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default Bus, Cluster, Server
Range Bus, Host

Temporary Queue Name Prefix:

The prefix to apply to the names of temporary queues. This name is a maximum of 12 characters.

Data type String

Default Provider topic connection factory settings

Use this panel to view or change the configuration properties of the selected JMS topic connection factory for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server. These configuration properties control how connections are created between the JMS provider and the service integration bus that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Topic Connection Factories** and click **New**. The following fields appear on the **General** tab.

Settings that have a default value display that appropriate value. Any settings that have fixed values have a drop down menu.

Name:

The name of the topic connection factory.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI Name:

The JNDI name that is used to match this topic connection factory definition to the deployment descriptor. This entry is a `resource-ref` name.

Data type String

User Name:

The **User Name** used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly. If this field is used, then the Properties field `UserName` is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

The connection factory **User Name** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly. If a user name and password are specified, then an authentication alias is created for the factory where the password is encrypted.

Data type String

Password:

The password used to create an encrypted. If you complete this field, then the Password field in the Properties box is ignored.

If you specify a value for the **User Name** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Bus Name:

The name of the bus to which the topic connection factory connects.

Data type String

Client Identifier:

The name of the client. This field is required for durable topic subscriptions.

Data type String

Nonpersistent Messaging Reliability:

The reliability applied to nonpersistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus** destination. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Persistent Message Reliability:

The reliability applied to persistent JMS messages sent using this connection factory.

If you want different reliability delivery options for individual JMS destinations, you can set this property to **As bus destination**. The reliability is then defined by the Reliability property of the bus destination to which the JMS destination is assigned.

Default ReliablePersistent

Range

None There is no message reliability for nonpersistent messages. If a nonpersistent message cannot be delivered, it is discarded.

Best effort nonpersistent
Messages are never written to disk, and are thrown away if memory cache overruns.

Express nonpersistent
Messages are written asynchronously to persistent storage if memory cache overruns, but are not kept over server restarts.

Reliable nonpersistent
Messages can be lost if a messaging engine fails, and can be lost under normal operating conditions.

Reliable persistent
Messages can be lost if a messaging engine fails, but are not lost under normal operating conditions.

Assured persistent
Highest degree of reliability where assured message delivery is supported.

As Bus destination
Use the delivery option configured for the bus destination.

Durable Subscription Home:

The name of the durable subscription home.

Data type String

Share durable subscriptions:

Controls whether or not durable subscriptions are shared across connections with members of a server cluster.

Normally, only one session at a time can have a TopicSubscriber for a particular durable subscription. This property enables you to override this behavior, to enable a durable subscription to have multiple simultaneous consumers.

Data type Selection list
Default In cluster
Range

In cluster
Allows sharing of durable subscriptions when connections are made from within a server cluster.

Always shared
Durable subscriptions can be shared across connections.

Never shared
Durable subscriptions are never shared across connections.

Read Ahead:

Controls the read-ahead optimization during message delivery.

Default	Default
Range	Default, AlwaysOn and AlwaysOff

Target:

The name of the Workload Manager target group containing the messaging engine.

Data type	String

Target Type:

The type of Workload Manager target group that contains the messaging engine.

Default	BusMember
Range	BusMember, Custom, ME

Target Significance:

The priority of significance for the target specified.

Default	Preferred
Range	Preferred, Required

Target Inbound Transport Chain:

The name of the protocol that resolves to a group of messaging engines.

Data type	String

Provider Endpoints:

The list of comma separated endpoints used to connect to a bootstrap server.

Type a comma-separated list of endpoint triplets with the syntax: host:port:protocol.

Example	`localhost:7777:BootstrapBasicMessaging` where `BootstrapBasicMessaging` corresponds to the remote protocol InboundBasicMessaging (JFAP-TCP/IP).
Default	• If the host name is not specified, then the default `localhost` is used as a default value. • If the port number is not specified, then `7276` is used as a default value. • If the chain name is not specified, a predefined chain, such as `BootstrapBasicMessaging`, is used as a default value.

Connection Proximity:

The proximity that the messaging engine should have to the requester.

Default	Bus
Range	Bus, Host, Cluster, Server

Temporary Topic Name Prefix:

The prefix to apply to the names of temporary topics. This name is a maximum of 12 characters.

Data type	String

Default Provider queue destination settings

Use this panel to view or change the configuration properties of the selected JMS queue destination for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Queue Destinations**. Click **New**. The following fields appear on the **General** tab.

Name:

The name of the queue destination factory. You must complete this field.

Data type	String

Description:

A description of this queue destination for administrative purposes within WebSphere Application Server.

Data type	String

JNDI Name:

The JNDI name used to match this definition to a deployment descriptor `resource-env-ref` name.

Data type	String

Queue Name:

The name of the queue.

Data type	String

Delivery Mode:

The delivery mode for messages sent to this destination.

Data type	String
Range	Application, Persistent or NonPersistent
Default	Application

Time to Live:

The default length of time from its dispatch time that a message sent to this destination should be retained by the system, where **0** indicates that time to live value does not expire. Value from the producer is used if the Time to Live field is not completed.

Data type	Integer
Units	Milliseconds

Priority:

The priority for messages sent to this destination. The value from the producer is used if not completed.

Data type	Integer
Range	0 to 9 with **0** as the lowest priority and **9** as the highest priority

Read Ahead:

Used to control read-ahead optimization during message delivery.

Data type	String
Range	AsConnection, AlwaysOn and AlwaysOff
Default	AsConnection

Default Provider topic destination settings

Use this panel to view or change the configuration properties of the selected JMS topic destination for use with the internal product Java Message Service (JMS) provider that is installed with WebSphere Application Server.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Default Provider**. Right-click **Topic Destinations**, and click **New**. The following fields appear on the **General** tab.

Name:

The name of the topic destination entry.

Data type	String

Description:

A description of the entry.

Data type	String

JNDI Name:

The JNDI name used to match this definition to a deployment descriptor `resource-env-ref` name.

Data type	String

Topic Space:

The name of the topic space. This field is required.

Data type	String
Default	DEFAULT_TOPIC_SPACE

Topic Name:

The name of the topic. This field is required.

Data type	String

Delivery Mode:

The default mode for messages sent to this destination.

Data type	String
Range	Application, Persistent or NonPersistent
Default	Application

Time to Live:

The default length of time from its dispatch time that a message sent to this destination should be retained by the system, where **0** indicates that time to live value does not expire. Value from the producer is used if not completed.

Data type	Long
Units	Milliseconds

Priority:

The priority for messages sent to this destination. Value from producer is used if not completed.

Data type	Integer
Range	0 to 9 with **0** as the lowest priority and **9** as the highest priority

Read Ahead:

Used to control read-ahead optimization during message delivery.

Data type	String
Range	AsConnection, AlwaysOn and AlwaysOff
Default	AsConnection

Version 5 Default Provider queue connection factory settings for application clients

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server version 5 applications. These configuration properties control how connections are created between the JMS provider and the default messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Provider** > **Version 5 Default Provider**. Right-click **Queue Connection Factories** and click **New**. The following fields appear on the **General** tab.

A queue connection factory is used to create JMS connections to queue destinations. The queue connection factory is created by the internal WebSphere Application Server product JMS provider. A Version 5 Default Provider queue connection factory has the following properties:

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The User ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a User ID and password explicitly, for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Re-Enter Password:

Confirms the password.

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type String

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the internal product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and the messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Topic Connection Factories** and click **New**. The following fields appear on the **General** tab.

A Version 5 Default Provider topic connection factory has the following properties.

Name:

The name by which this queue connection factory is known for administrative purposes within WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere Application Server administrative domain.

Data type String

Description:

A description of this topic connection factory for administrative purposes within WebSphere Application Server.

Data type String

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly, for example, if the calling application uses the method createTopicConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Node:

The WebSphere Application Server node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Range Pull-down list of nodes in the WebSphere Application Server administrative domain.

Application Server:

Enter the name of the application server. This name is not the host name of the machine, but the name of the configured application server.

Port:

Which of the two ports that connections use to connect to the JMS Server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for nonpersistent, nontransactional, nondurable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish or subscribe support. Therefore, any topic connection factory configured with the Port set to `Direct` cannot be used with message-driven beans.

Data type Enum
Default QUEUED

Range

QUEUED
The listener port used for full-function JMS compliant, publish or subscribe support.

DIRECT
The listener port used for direct TCP/IP connection (nontransactional, nonpersistent, and nondurable subscriptions only) for publish or subscribe support.

The TCP/IP port numbers for these ports are defined on the product internal JMS server.

Client ID:

The JMS client identifier used for connections to the MQSeries® queue manager.

Data type String

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Queue Destinations** and click **New**. The following fields are displayed on the **General** tab.

A queue destination is used to configure the properties of a JMS queue. A Version 5 Default Provider queue destination has the following properties.

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

Data type String

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or whether messages on the queue expire (have an unlimited expiry timeout).

Data type Enum

Default APPLICATION_DEFINED

Range

Application defined
The expiry timeout for messages in this queue is defined by the application that put them onto the queue.

Specified
The expiry timeout for messages in this queue is defined by the **Specified expiry** property.If you select this option, you must define a time out on the **Specified expiry** property.

Unlimited
Messages in this queue have no expiry timeout, and those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, specify the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer

Units Milliseconds

Range Greater than or equal to 0

- 0 indicates that messages never timeout.
- Other values are an integer number of milliseconds.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Version 5 Default Provider topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the internal product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **Version 5 Default Provider**. Right click **Topic Destinations** and click **New**. The following fields appear on the **General** tab.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. A Version 5 Default Provider topic has the following properties.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic, for administrative purposes within WebSphere Application Server.

Data type String

JNDI Name:

The application client run-time environment uses this field to retrieve configuration information.

Topic Name: The name of the topic as defined to the JMS provider.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range

Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.

Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Persistent
Messages on the destination are persistent.

Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range

Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.

Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

Specified
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, specify the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout).

Data type	Enum
Default	APPLICATION_DEFINED
Range	**Application defined** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **Specified** The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.* **Unlimited** Messages on this queue have no expiry timeout, so those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type	Integer
Units	Milliseconds
Range	Greater than or equal to 0 • 0 indicates that messages never time out. • Other values are an integer number of milliseconds.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere MQ Provider queue connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the MQSeries product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and WebSphere MQ.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right click **Queue Connection Factories**, and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see WebSphere MQ *Using Java*, available from the WebSphere MQ library.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

A queue connection factory for the JMS provider has the following properties.

Name:

The name by which this queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a userid and password explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly; for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Default Null

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the MQSeries queue manager for this connection factory.

Connections created by this factory connect to that queue manager.

Data type String

Host:

The name of the host on which the WebSphere MQ queue manager runs for client connection only.

Data type String
Default Null
Range A valid TCP/IP host name

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer
Default Null
Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.

Data type String
Default Null
Range 1 through 20 ASCII characters

Transport type:

Specifies whether the WebSphere MQ client connection or JNDI bindings are used for connection to the WebSphere MQ queue manager. The external JMS provider controls the communication protocols between JMS clients and JMS servers. Tune the transport type when you are using non-ASF nonpersistent, nondurable, nontransactional messaging or when you want to satisfy security issues and the client is local to the queue manager node.

Data type Enum
Units Not applicable

Default BINDINGS

Range

BINDINGS
JNDI bindings are used to connect to the queue manager. BINDINGS is a shared memory protocol and can only be used when the queue manager is on the same node as the JMS client and poses security risks that should be addressed through the use of EJB roles.

CLIENT
WebSphere MQ client connection is used to connect to the queue manager. CLIENT is a typical TCP-based protocol.

DIRECT
For WebSphere MQ Event Broker using DIRECT mode. DIRECT is a lightweight sockets protocol used in nontransactional, nondurable and nonpersistent Publish/Subscribe messaging. DIRECT only works for clients and message-driven beans using the non-ASF protocol.

QUEUED
QUEUED is a standard TCP protocol.

Recommended

Queue connection factory transport type
BINDINGS is faster by 30% or more, but it lacks security. When you have security concerns, BINDINGS is more desirable than CLIENT.

Topic connection factory transport type
DIRECT is the fastest type and should be used where possible. Use BINDINGS when you want to satisfy additional security tasks and the queue manager is local to the JMS client. QUEUED is the fallback for all other cases. WebSphere MQ 5.3 before CSD2 with the DIRECT setting can lose messages when used with message-driven beans and under load. This loss also happens with client-side applications unless the broker maxClientQueueSize is set to 0. You can set this to 0 with the command:

```
#wempschangeproperties WAS_nodeName_server1
 -e default -o DynamicSubscriptionEngine -n
 maxClientQueueSize -v 0 -x executionGroupUUID
```

where executionGroupUUID can be found by starting the broker and looking in the Event Log/Applications for event 2201. This value is usually ffffffff-0000-0000-000000000000.

Note: The WebSphere MQ 5.3 JMS cannot be used within WAS 6.1 because WAS 6.1 has a Java 5 runtime. Therefore, cross-memory connections cannot be established with WebSphere MQ 5.3 queue managers. This can result in a performance degradation if you were previously using WebSphere MQ 5.3 and BINDINGS for your connections and move to CLIENT network connections in migrating to WAS 6.1.

Client ID:

The JMS client identifier used for connections to the MQSeries queue manager.

Data type String

CCSID:

The coded character set identifier for use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ*

Application Programming Reference books. These references are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www.ibm.com/software/integration/wmq/library/, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Message Retention:

Select this check box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are handled according to their disposition options.

Data type Enum
Units Not applicable
Default Cleared
Range **Selected**
Unwanted messages are left on the queue.
Cleared
Unwanted messages are handled according to their disposition options.

Temporary model:

The name of the model definition used to create temporary connection factories if a connection factory does not already exist.

Data type String
Range 1 through 48 ASCII characters

Temporary queue prefix:

The prefix used for dynamic queue naming.

Data type String

Fail if quiesce:

Specifies whether applications return from a method call if the queue manager has entered a controlled failure.

Data type Check box
Default Selected

Local Server Address:

Specifies the local server address.

Data type String

Polling Interval:

Specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery

Data type Integer
Units Milliseconds

Default	5000

Rescan interval:

Specifies the interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	Milliseconds
Default	5000

SSL cipher suite:

Specifies the cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider. The value must match the CipherSpec specified on the SVRCONN channel as the **Channel** property.

You must set this property, if you set the **SSL Peer Name** property.

SSL certificate store:

Specifies a list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. If you specify a value for this property, you must use WebSphere MQ JVM at Java 2 version 1.4.

The value is a space-delimited list of entries of the form:
`ldap://`*`hostname`*`:[`*`port`*`]`

A single slash (/) follows this value. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connection time.

If this property is not set, such certificate checking is performed.

The SSL peer name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:
`CN=QMGR.*, OU=IBM, OU=WEBSPHERE`

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the section "Distinguished Names" in the WebSphere MQ Security book.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Data type	Check box
Default	Selected

WebSphere MQ Provider topic connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the WebSphere MQ product Java Message Service (JMS) provider. These configuration properties control how connections are created between the JMS provider and WebSphere MQ.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right-click **Topic Connection Factories** and click **New**.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product JMS resources. For more information about configuring WebSphere MQ product JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

MA0C broker: When creating a WebSphere Application Server v6 topic connection factory for the MA0C broker, you should consider the following attribute values:

BrokerControlQueue
: This value is fixed at SYSTEM.BROKER.CONTROL.QUEUE for the MA0C broker and is the queue the broker reads from.

BrokerVersion
: Set this value to BASIC for the MA0C broker.

ClientID
: Set this value to whatever you like for the MA0C broker (the value is string and is merely an identifier for your client application).

XA Enabled
Set this value to TRUE or FALSE for the MA0C broker (the setting you use is a performance enhancement flag - you will probably want to set this to 'true' most of the time).

BrokerMessage Selection
This value is fixed at CLIENT for the MA0C broker because the broker relies on client side message selection.

Direct Broker Authorization Type
This value is not required by the MA0C broker.

A topic connection factory for the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which this topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS provider.

Data type	String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String

JNDI Name:

The Java Naming and Directory Interface (JNDI) name that is used to bind the topic connection factory into the application server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String
Units	En_US ASCII characters
Range	1 through 45 ASCII characters

User ID:

The user ID used, with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User** property, you must also specify a value for the **Password** property.

The connection factory **User** and **Password** properties are used if the calling application does not provide a `userid` and `password` explicitly, for example, if the calling application uses the method createTopicConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used, with the **User ID** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String

Re-Enter Password:

Confirms the password.

Queue Manager:

The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.

Data type String

Host:

The name of the host on which the WebSphere MQ queue manager runs for client connections only.

Data type String
Range A valid TCP/IP host name

Port:

The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.

This port must be configured on the WebSphere MQ queue manager.

Data type Integer
Range A valid TCP/IP port number, configured on the WebSphere MQ queue manager.

Channel:

The name of the channel used for client connections to the WebSphere MQ queue manager for client connection only.

Data type String
Range 1 through 20 ASCII characters

Transport Type:

Whether WebSphere MQ client connection or JNDI bindings are used for connection to the WebSphere MQ queue manager.

Data type Enum
Default BINDINGS
Range
CLIENT
WebSphere MQ client connection is used to connect to the WebSphere MQ queue manager.
BINDINGS
JNDI bindings are used to connect to the WebSphere MQ queue manager.

Client ID:

The JMS client identifier used for connections to the WebSphere MQ queue manager.

Data type String

CCSID:

The coded character set identifier to be used with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

Data type String

Broker Control Queue:

The name of the broker control queue to which all command messages (except publications and requests to delete publications) are sent.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Queue Manager:

The name of the WebSphere MQ queue manager that provides the Publisher and Subscriber message broker.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Publish Queue:

The name of the broker input queue that receives all publication messages for the default stream.

The name of the broker's input queue (stream queue) that receives all publication messages for the default stream. Applications can also send requests to delete publications on the default stream to this queue.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Subscribe Queue:

The name of the broker queue from which nondurable subscription messages are retrieved.

The name of the broker queue from which nondurable subscription messages are retrieved. The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker CCSubQ:

The name of the broker queue from which nondurable subscription messages are retrieved for a ConnectionConsumer request. This property applies only for use of the Web container.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Broker Version:

Specifies whether the message broker is provided by the WebSphere MQ MA0C SupportPac™ or newer versions of WebSphere family message broker products.

Data type Enum
Default Advanced
Range
Advanced The message broker is provided by newer versions of WebSphere family message broker products (MQ Integrator and MQ Publish and Subscribe).
Basic The message broker is provided by the WebSphere MQ MA0C SupportPac (WebSphere MQ - Publish and Subscribe).

Cleanup level:

Specifies the level of clean up provided by the publish or subscribe cleanup utility.

Data type Enum
Default SAFE
Range
ASPROP
NONE
STRONG

Cleanup interval:

Specifies the interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Data type Integer

Units Milliseconds
Default 6000

Message selection:

Specifies where broker message selection is performed.

Data type Enum
Default BROKER
Range

BROKER
Message selection is done at the broker location.

Message CLIENT
Message selection is done at the client location.

Publish acknowledge interval:

The interval, in number of messages, between publish requests that require acknowledgement from the broker.

Data type Integer
Default 25

Sparse subscriptions:

Enables sparse subscriptions.

Data type Check box
Default Cleared

Status refresh interval:

The interval, in milliseconds, between transactions to refresh publish or subscribe status.

Data type Integer
Default 6000

Subscription store:

Specifies where WebSphere MQ stores data relating to active JMS subscriptions.

Data type Enum
Default MIGRATE
Range

MIGRATE

QUEUE

BROKER

Multicast:

Specifies whether this connection factory uses multicast transport.

Data type Enum
Default NOT USED
Range

NOT USED
This connection factory does not use multicast transport.

ENABLED
This connection factory always uses multicast transport.

ENABLED_IF_AVAILABLE
This connection factory uses multicast transport.

ENABLED_RELIABLE
This connection factory uses reliable multicast transport.

ENABLED_RELIABLE_IF_AVAILABLE
This connection factory uses reliable multicast transport if available.

Direct authentication:

Specifies whether to use direct broker authorization.

Data type Enum
Default NONE
Range

NONE Direct broker authorization is not used.

PASSWORD
Direct broker authorization is authenticated with a password.

CERTIFICATE
Direct broker authorization is authenticated with a certificates.

Proxy Host Name:

Specifies the host name of a proxy to be used for communication with WebSphere MQ.

Data type String

Proxy Port:

Specifies the port number of a proxy to be used for communication with WebSphere MQ.

Data type Integer
Default 0

Fail if quiesce:

Specifies whether applications return from a method call if the queue manager has entered a controlled failure.

Data type Check box
Default Selected

Local Server Address:

Specifies the local server address.

Data type	String

Polling Interval:

Specifies the interval, in milliseconds, between scans of all receivers during asynchronous message delivery.

Data type	Integer
Units	Milliseconds
Default	5000

Rescan interval:

Specifies the interval in milliseconds between which a topic is scanned to look for messages that have been added to a topic out of order.

This interval controls the scanning for messages that have been added to a topic out of order with respect to a WebSphere MQ browse cursor.

Data type	Integer
Units	Milliseconds
Default	5000

SSL cipher suite:

Specifies the cipher suite to use for SSL connection to WebSphere MQ.

Set this property to a valid cipher suite provided by your JSSE provider. The value must match the CipherSpec specified on the SVRCONN channel as the **Channel** property.

You must set this property, if you set the **SSL Peer Name** property.

SSL certificate store:

Specifies a list of zero or more Certificate Revocation List (CRL) servers used to check for SSL certificate revocation. If you specify a value for this property, you must use WebSphere MQ JVM at Java 2 version 1.4.

The value is a space-delimited list of entries of the form:

`ldap://`*`hostname`*`:[`*`port`*`]`

A single slash (/) follows this value. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. For more information about CRL security, see the section “Working with Certificate Revocation Lists” in the *WebSphere MQ Security book*; for example at: http://publibfp.boulder.ibm.com/epubs/html/csqzas01/csqzas012w.htm#IDX2254.

SSL peer name:

For SSL, a *distinguished name* skeleton that must match the name provided by the WebSphere MQ queue manager. The distinguished name is used to check the identifying certificate presented by the server at connection time.

If this property is not set, such certificate checking is performed.

The SSL peer name property is ignored if **SSL Cipher Suite** property is not specified.

This property is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
CN=QMGR.*, OU=IBM, OU=WEBSPHERE
```

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

For more details about distinguished names and their use with WebSphere MQ, see the section "Distinguished Names" in the WebSphere MQ Security book.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. This is independent from any WebSphere MQ connection pooling. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Data type	Check box
Default	Selected

WebSphere MQ Provider queue destination settings for application clients

Use this panel to view or change the configuration properties of the selected queue destination for use with the WebSphere MQ product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right-click **Queue Destinations** and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product for JMS resources. For more information about configuring WebSphere MQ product for JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters.

A queue for use with the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

Data type String

JNDI Name:

The application client run-time environment uses this field to retrieve configuration information.

Persistence:

Whether all messages sent to the destination are persistent, nonpersistent or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them onto the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them onto the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified Priority:

If the **Priority** property is set to `Specified`, specify the message priority for this queue, in the range 0 (lowest) through 9 (highest).

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout value for this queue is defined by the application or the by **Specified expiry** property or whether messages on the queue never expire (have an unlimited expiry time out).

Data type Enum
Units Not applicable
Default APPLICATION_DEFINED
Range

Application defined
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

Specified
The expiry timeout for messages on this queue is defined by the **Specified expiry** property. If you select this option, you must define a timeout on the **Specified expiry** property.

Unlimited
Messages on this queue have no expiry timeout and those messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer
Units Milliseconds
Range Greater than or equal to 0
- 0 indicates that messages never time out
- Other values are an integer number of milliseconds

Base Queue Name:

The name of the queue to which messages are sent, on the queue manager specified by the **Base queue manager name** property.

Data type String

Base Queue Manager Name:

The name of the WebSphere MQ queue manager to which messages are sent.

This queue manager provides the queue specified by the **Base queue name** property.

Data type String

Units En_US ASCII characters
Range A valid WebSphere MQ Queue Manager name, as 1 through 48 ASCII characters

CCSID:

The coded character set identifier to use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSID identifier supported by WebSphere MQ queue manager.

Data type String

Integer encoding:

If native encoding is not enabled, select whether integer encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

Indicates that if native encoding is not enabled to select whether decimal encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

Indicates that if native encoding is not enabled to select the type of floating point encoding.

Data type Enum
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S390 floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Indicates that the queue destination use native encoding (appropriate encoding values for the Java platform) when you select this check box.

Data type Enum
Default Cleared
Range
Cleared
Native encoding is not used, so specify the following properties for integer, decimal and floating point encoding.
Selected
Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Target client:

Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application.

Data type Enum
Default WebSphere MQ
Range
WebSphere MQ
The target is a traditional WebSphere MQ application that does not support JMS.
JMS The target application supports JMS.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

WebSphere MQ Provider topic destination settings for application clients

Use this panel to view or change the configuration properties of the selected topic destination for use with the WebSphere MQ product Java Message Service (JMS) provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > **WebSphere MQ Provider**. Right click **Topic Destinations**, and click **New**. The following fields are displayed on the **General** tab.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ product JMS resources. For more information about configuring WebSphere MQ product JMS resources, see the WebSphere MQ *Using Java* book.
- In WebSphere MQ, names can have a maximum of 48 characters.

A topic destination is used to configure the properties of a JMS topic for the associated JMS provider. A topic for use with the WebSphere MQ product JMS provider has the following properties.

Name:

The name by which the topic is known for administrative purposes.

Data type String

Description:

A description of the topic for administrative purposes within IBM WebSphere Application Server.

JNDI Name:

The application client run time uses this field to retrieve configuration information.

Persistence:

Specifies whether all messages sent to the destination are persistent, nonpersistent, or have their persistence defined by the application.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
Messages on the destination have their persistence defined by the application that put them in the queue.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Persistent
Messages on the destination are persistent.
Nonpersistent
Messages on the destination are not persistent.

Priority:

Specifies whether the message priority for this destination is defined by the application or the **Specified priority** property.

Data type Enum
Default APPLICATION_DEFINED
Range
Application defined
The priority of messages on this destination is defined by the application that put them in the destination.
Queue defined
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. If you select this option, you must define a priority for the **Specified priority** property.

Specified Priority:

If the **Priority** property is set to `Specified`, type the message priority for this queue, in the range 0 (lowest) through 9 (highest).

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or by the **Specified expiry** property or by messages on the queue never expire (have an unlimited expiry timeout).

Data type Enum
Default APPLICATION_DEFINED
Range

Application defined
The expiry timeout for messages on this queue is defined by the application that put them in the queue.

Specified
The expiry timeout for messages in this queue is defined by the **Specified expiry** property. If you select this option, you must define a timeout value for the **Specified expiry** property.

Unlimited
Messages on this queue have no expiry timeout, and these messages never expire.

Specified Expiry:

If the **Expiry timeout** property is set to `Specified`, type the number of milliseconds (greater than 0) after which messages on this queue expire.

Data type Integer
Units Milliseconds
Range Greater than or equal to 0

- 0 indicates that messages never time out.
- Other values are an integer number of milliseconds.

Base Topic Name:

The name of the topic to which messages are sent.

Data type String

CCSID:

The coded character set identifier to use with the WebSphere MQ queue manager.

This coded character set identifier (CCSID) must be one of the CCSID identifiers that WebSphere MQ supports.

Data type String

Units Integer
Range 1 through 65535

Integer encoding:

Indicates whether integer encoding is normal or reversed when native encoding is not enabled.

Data type Enum
Default NORMAL
Range
NORMAL
Normal integer encoding is used.
REVERSED
Reversed integer encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Decimal encoding:

If native encoding is not enabled, select whether decimal encoding is normal or reversed.

Data type Enum
Default NORMAL
Range
NORMAL
Normal decimal encoding is used.
REVERSED
Reversed decimal encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Floating point encoding:

Indicates the type of floating point encoding when native encoding is not enabled.

Data type Enum
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
S390 S/390® floating point encoding is used.

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

Native encoding:

Indicates that the queue destination uses native encoding (appropriate encoding values for the Java platform) when you select this check box.

Data type Enum
Default Cleared

Range

Cleared
Native encoding is not used, so specify the previous properties for integer, decimal and floating point encoding.

Selected
Native encoding is used (to provide appropriate encoding values for the Java platform).

For more information about encoding properties, see the WebSphere MQ *Using Java* document.

BrokerDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved.

The subscriber specifies the name of the queue when it registers a subscription.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

BrokerCCDurSubQueue:

The name of the broker queue from which durable subscription messages are retrieved for a ConnectionConsumer. This property applies only for use of the Web container.

Data type String
Units En_US ASCII characters
Range 1 through 48 ASCII characters

Target Client:

Specifies whether the receiving application is JMS compliant or is a traditional WebSphere MQ application.

Data type Enum
Default WebSphere MQ
Range

WebSphere MQ
The target is a traditional WebSphere MQ application that does not support JMS.

JMS The target is a JMS compliant application.

Multicast:

Specifies whether this connection factory uses multicast transport.

Data type Enum
Default AS_CF

Range

AS_CF This connection factory uses multicast transport.

DISABLED
This connection factory does not use multicast transport.

NOT_RELIABLE
This connection factory always uses multicast transport.

RELIABLE
This connection factory uses multicast transport when the topic destination is not reliable.

ENABLED
This connection factory uses reliable multicast transport.

Generic JMS connection factory settings for application clients

Use this panel to view or change the configuration properties of the selected Java Message Service (JMS) connection factory for use with the associated JMS provider. These configuration properties control how connections are created between the JMS provider and the messaging system that it uses.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > *new_JMS_Provider_instance*. Right-click **Connection Factories**, and click **New**. The following fields are displayed on the **General** tab.

A Java Message Service (JMS) connection factory creates connections to JMS destinations. The JMS connection factory is created by the associated JMS provider. A JMS connection factory for a generic JMS provider (other than the internal default messaging provider or WebSphere MQ as a JMS provider) has the following properties:

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated JMS provider.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

JNDI Name:

The application client run time uses this field to retrieve configuration information.

User ID:

Indicates the user ID used with the **Password** property, for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

The connection factory **User ID** and **Password** properties are used if the calling application does not provide a userid and password explicitly; for example, if the calling application uses the method createQueueConnection(). The JMS client flows the `userid` and `password` to the JMS server.

Data type String

Password:

The password used with the **User ID** property for authentication if the calling application does not provide a `userid` and `password` explicitly.

If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.

Data type String
Default Null

Re-Enter Password:

Confirms the password entered in the **Password** field.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server name space.

As a convention, use the fully qualified JNDI name, for example, `jms/`*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI API by the platform.

Data type String

Connection Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publication or subscription).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish subscribe messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Generic JMS destination settings for application clients

Use this panel to view or change the configuration properties of the selected JMS destination for use with the associated JMS provider.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Messaging Providers** > *new JMS Provider instance*. Right-click **Destinations**, and click **New**. The following fields are displayed on the **General** tab.

A JMS destination is used to configure the properties of a JMS destination for the associated generic JMS provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS destination for use with a generic JMS provider (not the default messaging provider or WebSphere MQ as a JMS provider) has the following properties.

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Description:

A description of the queue, for administrative purposes.

JNDI Name:

The JNDI name of the actual (physical) name of the JMS destination bound into JNDI.

External JNDI Name:

The JNDI name that is used to bind the queue into the application server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Destination Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publishing or subscribing).

Select one of the following options:

Queue
A JMS queue destination for point-to-point messaging.

Topic A JMS topic destination for pub/sub messaging.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Example: Configuring JMS provider, JMS connection factory and JMS destination settings for application clients

You can configure JMS Provider, JMS Connection Factory and JMS Destination settings. This topic provides the required fields, special cases, and an example.

The purpose of this article is to help you to configure JMS Provider, JMS Connection Factory and JMS Destination settings.

- Required fields include:
  - JMS Provider Properties page: name, and at least one protocol provider
  - JMS Connection Factory Properties page: name, jndiName, destination type
  - JMS Destination Properties page: name, jndiName, destination type
- Special cases:
  - The destination type must be `QUEUE`, or `TOPIC`.
- Example:

```
<resources.jms:JMSProvider xmi:id="JMSProvider_3" name="genericJMSProvider:name"
description="genericJMSProvider:description"
externalInitialContextFactory="genericJMSProvider:contextFactoryClass"
externalProviderURL="genericJMSProvider:providerUrl">
<classpath>genericJMSProvider:classpath</classpath>
<factories xmi:type="resources.jms:GenericJMSDestination"
xmi:id="GenericJMSDestination_1" name="jmsDestination:name"
jndiName="jmsDestination:jndiName" description="jmsDestination:description"
externalJNDIName="jmsDestination:externalJndiName" type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_15">
<resourceProperties xmi:id="J2EEResourceProperty_17" name="jmsDestination:customName"
value="jmsDestination:customValue"/>
</propertySet>
</factories>
<factories xmi:type="resources.jms:GenericJMSConnectionFactory"
xmi:id="GenericJMSConnectionFactory_1" name="jmsCF:name" jndiName="jmsCF:jndiName"
description="jmsCF:description" userID="jmsCF:user" password="{xor}NTIsHBllMT4yOg=="
externalJNDIName="jmsCF:externalJndiName" type="QUEUE">
<propertySet xmi:id="J2EEResourcePropertySet_16">
<resourceProperties xmi:id="J2EEResourceProperty_18" name="jmsCF:customName"
value="jmsCF:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_17">
<resourceProperties xmi:id="J2EEResourceProperty_19"
name="genericJMSProvider:customName" value="genericJMSProvider:customValue"/>
</propertySet>
</resources.jms:JMSProvider>
```

Configuring new JMS connection factories for application clients

Use this task to create a new Java Message Service (JMS) connection factory configuration for your application client.

1. Click the JMS provider for which you want to create a connection factory in the tree. Complete one of the following actions:
   - Configure a new JMS provider.
   - Click an existing JMS provider.
2. Expand the JMS provider to view its **Connection Factories** folder.
3. Click the connection factory folder, and complete one of the following actions:
   - Right-click the folder and select**New**.
   - Click **Edit > New** on the menu bar.
4. Configure the JMS connection factory properties in the displayed fields.

5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring new JMS destinations for application clients

Use this task to create a new Java Message Service (JMS) destination configuration for your application client.

1. Click the JMS provider in the tree for which you want to create a destination. Complete one of the following actions:
   - Configure a new JMS provider.
   - Click an existing JMS provider.
2. Expand the JMS provider to view its **Destinations** folder.
3. Click the provider folder, and complete one of the following actions:
   - Right-click the folder and select **New**.
   - Click **Edit > New** on the menu bar.
4. Configure the JMS destination properties in the displayed fields.
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Configuring new resource environment providers for application clients

You can create new resource environment provider configurations for your application client using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

During this task, you create new resource environment provider configurations for your application client.

About this task

To configure a new resource environment provider, perform the following steps:

1. Start the Application Configuration Resource Tool and open the EAR file for which you want to configure the new Java Message Service (JMS) provider. The EAR file contents display in a tree view.
2. Select from the tree the JAR file in which you want to configure the new JMS provider.
3. Expand the JAR file to view its contents.
4. Click the **Resource Environment Providers** folder. Take one of the following actions:
   - Right-click the provider folder, and click **New**.
   - Click **Edit > New** on the menu bar.
5. Configure the JMS provider properties in the displayed fields.
6. Click **OK** when you finish.
7. Click **File > Save** on the menu bar to save your changes.

Resource environment provider settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected Java Archive (JAR) file. Right-click **Resource Environment Providers**, and click **New**. The following fields are displayed on the **General** tab:

Name:

Specifies the administrative name for the resource environment provider.

Description:

Specifies a description of the resource environment provider for your administrative records.

Class Path:

Specifies the path to the JAR file that contains the implementation classes for the resource environment provider.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Configuring new resource environment entries for application clients

You can create new resource environment entries for your client application using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

During this task, you create new resource environment entries for your client application.

About this task

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the EAR file for which you want to configure the new resource environment entry. The EAR file contents are in the displayed tree view.
3. Click the desired resource environment provider, and complete the following action to configure new providers:
   - Configure a new resource environment provider.
4. Expand the resource environment provider to view the **Resource Environment Entries** folder.
5. Click the resource environment entries folder, and complete one of the following actions:
   - Right-click the folder and select **New**.
   - Click **Edit > New** on the menu bar.
6. Configure the resource environment entry properties in the displayed fields.
7. Click **OK**.
8. Click **File > Save** on the menu bar to save your changes.

Resource environment entry settings for application clients

Use this page to specify resource environment entry properties.

To view this Application Client Resource Configuration Tool (ACRCT) page, click **File** > **Open**. After you browse for an EAR file, click **Open**. Expand the selected JAR file > **Resource Environment Providers** > *resource environment instance*. Right-click **Resource Environment Entries**, and click **New**. The following fields appear on the **General** tab:

Name:

Specifies the administrative name for the resource environment entry.

Description:

Specifies a description of the URL for your administrative records.

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

Use this name to link to the binding information of the platform. The binding associates the resources defined in the deployment descriptor of the module to the actual (or physical) resources bound into JNDI by the platform.

Custom Properties:

Specifies name-value pairs for setting additional properties on the object that is created at run time for this resource.

You must enter a name that is a public property on the object and a value that can be converted from a string to the type required by the set method of the property. The acceptable properties and values depend on the object that is created. Refer to the object documentation for a list of valid properties and values.

Managing application clients

You can manage Java Platform, Enterprise Edition (Java EE) application clients using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

Perform the following tasks after deploying application clients. This task only applies to Java EE application clients.

1. Update data source and data source provider configurations.
2. Update URLs and URL provider configurations.
3. Update mail session configurations.
4. Update JMS provider, connection factories, and destination configurations.
5. Update MQ JMS provider, MQ connection factories and MQ destination configurations.
6. Update Resource Environment Entry and Resource Environment Provider configurations.
7. (Optional) Remove application client resources.

Updating data source and data source provider configurations with the Application Client Resource Configuration Tool

You can update the configuration of an existing data source or data source provider using the Application Client Resource Configuration Tool (ACRCT).

About this task

During this task, you update the configuration of an existing data source or data source provider. Perform this task when your database configuration changes.

1. Start the Application Client Resource Configuration Tool (ACRCT), and open the Enterprise Archive (EAR) file containing the data source or data source provider. The EAR file contents display in a tree view.
2. Select Java Archive (JAR) file from the navigation tree containing the data source or data source provider to update.
3. Expand the JAR file to view its contents until you locate the particular data source or data source provider to update. Take one of the following actions:
   - Right-click the data source object and click **Properties**.

- Click **Edit > Properties** on the menu bar.

4. Update the properties in the displayed fields. For detailed field help, see:
   - Data source provider properties
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Updating URLs and URL provider configurations for application clients

You can update URLs and URL provider configurations for application clients using the Application Client Resource Configuration Tool (ACRCT).

1. Start the tool and open the Enterprise Archive (EAR) file containing the URL or URL provider. The EAR file contents are displayed in a tree view.
2. Select from the tree the Java Archive (JAR) file containing the URL or URL provider to update.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular URL or URL provider to update. Take one of the following actions:
   a. Right-click the URL object and click **Properties**.
   b. Click **Edit** > **Properties** on the menu bar.
5. Update the properties in the displayed fields.
6. Click **OK** when you finish.
7. Click **File** > **Save** on the menu bar to save your changes.

Updating mail session configurations for application clients

You can update the configuration of an existing JavaMail session using the Application Client Resource Configuration Tool (ACRCT).

About this task

During this task, you update the configuration of an existing JavaMail session. You cannot update the name of the default JavaMail provider, and you cannot delete the default JavaMail provider from the navigation tree.

1. Start the tool and open the Enterprise Archive (EAR) file containing the JavaMail session. The EAR file contents are displayed in the navigation tree view.
2. Select the Java Archive (JAR) file containing the JavaMail session to update from the navigation tree.
3. Expand the JAR file to view its contents.
4. Keep expanding the JAR file contents until you locate the particular JavaMail session to update. Take one of the following actions:
   a. Right-click the object and click **Properties**
   b. Click **Edit** > **Properties** from the menu bar.
5. Update the properties in the displayed fields.
6. Click **OK** when you finish.
7. Select **File** > **Save** from the menu bar to save your changes.

Updating Java Message Service provider, connection factories, and destination configurations for application clients

You can update the configuration of an existing Java Message Service (JMS) provider, connection factory or destination using the Application Client Resource Configuration Tool (ACRCT).

About this task

During this task, you update the configuration of an existing Java Message Service (JMS) provider, connection factory or destination.

1. Start the tool and open the Enterprise Archive (EAR) file containing the Java Message Service (JMS) provider, connection factory, or destination. The EAR file contents display in a tree view.
2. Select the Java Archive (JAR) file containing the JMS provider, connection factory, or destination to update from the navigation tree.
3. Expand the JAR file to view its contents until you locate the particular JMS provider, connection factory, or destination to update. When you find it, do one of the following actions:
   - Right-click the provider, and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
4. Update the properties in the displayed fields. For detailed field help, see:
   - JMS provider properties
   - WebSphere Application Server Queue connection factory properties
   - WebSphere Application Server Topic connection factory properties
   - WebSphere Application Server Queue destination properties
   - WebSphere Application Server Topic destination properties
5. Click **OK**.
6. Click **File > Save** to save your changes.

Updating WebSphere MQ as a Java Message Service provider, and its JMS resource configurations, for application clients

You can update an existing configuration of WebSphere MQ as a Java Message Service (JMS) provider, and update the configuration of WebSphere MQ connection factories or WebSphere MQ destinations.

About this task

Use this task to update an existing configuration of WebSphere MQ as a Java Message Service (JMS) provider, and to update the configuration of WebSphere MQ connection factories or WebSphere MQ destinations.

1. Start the Application Client Resource Configuration Tool (ACRCT).
2. Open the Enterprise Archive (EAR) file containing the WebSphere MQ JMS provider, WebSphere MQ connection factory, or WebSphere MQ destination. The EAR file contents are displayed in the navigation tree view.
3. Select the Java Archive (JAR) file containing the JMS provider, connection factory, or destination to update.
4. Expand the JAR file to view its contents until you locate the particular JMS provider, connection factory, or destination that you want to update. Complete one of the following actions:
   - Right-click the appropriate object and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
5. Update the properties in the displayed fields. For detailed field help, see:
   - JMS provider properties
   - MQ Queue connection factory properties
   - MQ Topic connection factory properties
   - MQ Queue destination properties
   - MQ Topic destination properties
6. Click **OK**.
7. Click **File > Save** to save your changes.

Updating resource environment entry and resource environment provider configurations for application clients

You can update the configuration of an existing resource environment entry or resource environment provider using the Application Client Resource Configuration Tool (ACRCT).

About this task

During this task, you update the configuration of an existing resource environment entry or resource environment provider.

1. Start the tool and open the Enterprise Archive (EAR) file containing the resource environment entry or resource environment provider. The EAR file contents display in a navigation tree view.
2. Select from the tree the Java Archive (JAR) file containing the resource environment entry or resource environment provider to update.
3. Expand the JAR file to view its contents until you locate the resource environment entry or resource environment provider to update. Take one of the following actions:
   - Right-click the resource environment object, and click **Properties**.
   - Click **Edit > Properties** on the menu bar.
4. Update the properties in the displayed fields. For detailed field help, see:
   - Resource environment provider properties
   - Resource environment entry properties
5. Click **OK** when you finish.
6. Click **File > Save** on the menu bar to save your changes.

Example: Configuring Resource Environment settings:

You can configure Resource Environment settings. This topic provides the required fields and an example.

The purpose of this topic is to help you configure Resource Environment settings.

- Required fields:
  - Resource Environment Provider page: **Name**
  - Resource Environment Entry page: **Name, JNDI Name**
- Example:

```
<resources.env:ResourceEnvironmentProvider xmi:id="ResourceEnvironmentProvider_1"
name="resourceEnvProvider:name" description="resourceEnvProvider:description">
<classpath>resourceEnvProvider:classpath</classpath>
<factories xmi:type="resources.env:ResourceEnvEntry" xmi:id="ResourceEnvEntry_1"
name="resourceEnvEntry:name" jndiName="resourceEnvEntry:jndiName"
description="resourceEnvEntry:description">
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22"
name="resourceEnvEntry:customName" value="resourceEnvEntry:customValue"/>
</propertySet>
</factories>
<propertySet xmi:id="J2EEResourcePropertySet_21">
<resourceProperties xmi:id="J2EEResourceProperty_23"
name="resourceEnvProvider:customName" value="resourceEnvProvider:customValue"/>
</propertySet>
</resources.env:ResourceEnvironmentProvider>
```

Example: Configuring resource environment custom settings for application clients:

You can configure resource environment custom settings.

The purpose of this topic is to help you configure resource environment custom settings.

- The custom page applies to every resource type. You can specify as many custom names and values as you need.
- Example:

```
<propertySet xmi:id="J2EEResourcePropertySet_20">
<resourceProperties xmi:id="J2EEResourceProperty_22"
name="resourceEnvEntry:customName" value="resourceEnvEntry:customValue"/>
</propertySet>
```

Removing application client resources

You can remove Java Platform, Enterprise Edition (Java EE) application client resources using the Application Client Resource Configuration Tool (ACRCT).

Before you begin

The option to delete an item does not offer a confirmation dialog. As a safeguard, consider saving your work right before you begin this task. If you change your mind after removing an item, you can close the EAR file without saving your changes, canceling your deletion. Remember to close the EAR file immediately after the deletion, or you also lose any unsaved work that you performed since the deletion.

This task only applies to Java EE application clients.

1. Start the Application Client Resource Configuration Tool (ACRCT) and open the Enterprise Archive (EAR) file from which you want to remove an object. The EAR file contents display in the navigation tree view. If you already have an EAR file open and have made some changes, click **File** > **Save** to save your work before preceding to delete an object.
2. Locate the object that you want to remove in the tree.
3. Right-click the object, and click **Delete**.
4. Click **File** > **Save**.

Chapter 5. Web services

Task overview: Implementing Web services applications

Use this topic as an introduction to using Web services. WebSphere Application Server supports Web services that are developed and implemented based on a variety of Java programming models. Use Web services when operating across a variety of platforms, including Java Platform, Enterprise Edition (Java EE) and non-Java EE platforms.

Before you begin

Decide if a Web services implementation benefits your business process.

About this task

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Implementing Web services applications is an easy way to integrate application systems together within or outside your business infrastructure that function as stand-alone systems. For example, your customer information database is a stand-alone application, but you want your accounting application to access the customer data. You can create a Web service for the customer database and then enable the accounting application as a Web service client. The accounting application can now access the customer information. By implementing a Web service, these two applications can share information in an efficient way.

Because Web services are easily applied to existing applications and information technology assets, you can develop, deploy and recompose new solutions quickly to address new opportunities. As Web services become more popular, the pool of services grows, promoting development of more robust models of just-in-time application and business integration over the Internet.

You can use Web services applications with the application server by following the steps provided:

1. Plan to use Web services. Review all of the components of Web services to learn how you can make your Web services plan more robust.
2. (Optional) Migrate existing Web services.

   Because Java EE environments emphasize compatibility, most application servers that offer support for the newer JAX-WS and JAXB specifications continue to support the older JAX-RPC specification. A consequence of this is that, existing Web services are likely to remain JAX-RPC based while new ones are developed using JAX-WS and JAXB.

   However, as time passes and applications are revised and rewritten, there might be times when the best strategy is to migrate a JAX-RPC based Web service to one based on JAX-WS and JAXB. This might result from a vendor choosing to provide enhancements to qualities of service that are only available in the new programming models. For example, SOAP 1.2 and SOAP Message Transmission Optimization Mechanism (MTOM) support are only available within the JAX-WS 2.x and JAXB 2.x

programming models and not JAX-RPC. Read about Web services migration best practices to learn more about best practices and examples when migrating JAX-RPC Web services to JAX-WS and JAXB Web services.

Note: Existing JAX-RPC applications wanting to use JAX-WS features must be rewritten using the JAX-WS programming model.

If you have used Web services based on Apache SOAP and now want to develop and implement Web Services for Java EE specification, you need to migrate client applications developed with all versions of 4.0, and versions of 5.0 prior to 5.0.2. See the topic, Migrating Apache SOAP Web services to JAX-RPC Web Services based on Java EE standards.

3. Develop Web services applications. You can develop Web services in one of the following ways:
   a. Develop Web services from existing WSDL files using JAX-WS.

      You can create a JAX-WS Web service by starting with an existing Web Services Description Language (WSDL) file describing the service interface for a JavaBeans or enterprise beans application. Typically, the WSDL file is defined as part of the application modeling process. Using an existing service definition or WSDL file to generate a new application is called a top-down approach to developing Web services.

   b. Develop Web services applications using JAX-WS.

      You can use the Java API for XML-Based Web Services (JAX-WS) programming model to develop Web services. JAX-WS simplifies application development through a standard, annotation-based model to develop Web services applications and clients. A common set of binding rules for XML and Java objects make it easy to incorporate XML data and process functions in Java applications. A further set of enhancements help you optimally send binary attachments, such as images or files, with the Web services requests.

      When developing a JAX-WS Web service starting from existing JavaBeans or stateless session enterprise beans, you can expose the bean as a JAX-WS Web service by using annotations. Adding the @WebService or @WebServiceProvider annotation to the bean defines the bean as a JAX-WS Web service. Enterprise beans that are exposed as JAX-WS Web services must be packaged in EJB 3.0 or higher modules.

      Transforming an existing application into Web services is called a bottoms-up approach to developing Web services. This process is called bottoms-up because you are starting with the implementation rather than starting with an existing service or Web Services Description Language (WSDL) file.

   c. Develop and deploy JAX-WS Web services clients Web services clients that can both access and invoke JAX-WS Web services are developed based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification. The application server supports Enterprise JavaBeans™ (EJB) clients, Java EE application clients, JavaServer Pages (JSP) files and servlets that are based on the JAX-WS programming model.

   d. Develop Web services applications from existing WSDL files with JAX-RPC.

      You can create a JAX-RPC Web service by starting with an existing WSDL file describing the service interface of an enterprise bean implementation using a top-down approach to developing Web services.

   e. Develop Web services applications with JAX-RPC.

      You can use the Java API for XML-based RPC (JAX-RPC) programming model to develop Web services. When developing a JAX-RPC Web service starting from existing JavaBeans or enterprise beans, you need develop a WSDL file. You can use existing JavaBeans or enterprise beans and then enable the implementation for Web services.

   f. Develop and deploy JAX-RPC Web services clients You can develop Web services clients based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification and the Java API for XML-based remote procedure call (JAX-RPC) specification. The application server supports Enterprise JavaBeans™ (EJB) clients, Java EE application clients, JavaServer Pages (JSP) files and servlets that are based on the JAX-RPC programming model.

g. Enable Web services through service integration technologies

   You can use the Web services enablement of the service integration bus (SIBus) to achieve the following goals:

   - Make an internally hosted service that is available at a bus destination available as a Web service.
   - Make an external Web service available internally at a bus destination.
   - Use the Web services gateway to map an existing service, either an internally-hosted service or an external Web service, to a new Web service that is provided by the gateway.

You can develop Web services to take advantage of Web Services Addressing (WS-Addressing), Web Services Resource Framework (WSRF), and Web Services Transaction (WS-Transaction) support.

- Use the WS-Addressing SPI: Performing more advanced Web Service Addressing tasks.

  You can develop Web services to take advantage of Web Services Addressing (WS-Addressing), which aids interoperability between Web services using a standard way to address Web services and providing addressing information in messages.
- Create stateful Web services using the Web Services Resource Framework.

  With the Web Services Resource Framework (WSRF) support in the application server, you can implement a stateful Web service as a WS-Resource, and reference that service it using a WS-Addressing endpoint reference.
- Use WS-Transaction policy to coordinate transactions or business activities for Web services.

  WS-Transaction is an interoperability standard that includes the WS-AtomicTransaction, WS-BusinessActivity, and WS-Coordination specifications. The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the Web services environment. Distributed Web services applications, and the resources they use, can take part in distributed global transactions. With Web Services Business Activity (WS-BA) support in the application server, Web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs.
- Use WS-Policy to exchange policies in a standard format.

  WS-Policy is an interoperability standard that is used to describe and communicate the policies of a Web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction.

4. Assemble Web services.

   Read about what you need to assemble a Web service and the order in which to assemble the parts, for example an enterprise archive (EAR) file.
5. Deploy Web services.

   Read about the steps necessary to deploy the EAR file that has been configured and enabled for Web services.
6. Administer deployed Web services.

   Once your Web services application is deployed, you can configure security settings, view deployment descriptors and WSDL documents, set the scope of a Web service port, and manage policy sets and service providers. These tasks can be done using the administrative console or with command-line tools.
7. Secure Web services.

   - Manage policy sets using the administrative console.

     Read about creating policy sets that you can use to simplify the security configuration of your Web services applications. Policy sets are only supported by JAX-WS applications.
   - Secure Web services applications using message level security.

Consider a broad set of security requirements, including authentication, authorization, privacy, trust, integrity, confidentiality, secure communications channels, delegation, and auditing across a spectrum of application and business topologies.

8. Publish the WSDL file.

   After installing a Web services application, and optionally modifying the endpoint information, you might need Web Services Description Language (WSDL) files containing the updated endpoint information. Read about the steps necessary to publish the WSDL files so that this information is available.

9. Monitor the performance of Web services applications.

   Read about using the Performance Monitoring Infrastructure (PMI) to measure the time required to process Web services requests.

10. Troubleshoot Web services.

    Read about troubleshooting different processes used to develop, implement and use Web services, including command-line tools, Java compiling errors, client runtime errors and exceptions, serialization and deserialization errors, and authentication challenges and authorization failures with Web services security.

Example

The following example illustrates how a business might use Web services.

The owner of a flower shop wants to start receiving orders from customers through the Web. This owner starts the process by finding wholesale flower suppliers, pricing the product, and completing contracts for future flower orders.

Using Web services, the flower shop owner can find wholesale flower suppliers. One way to find new suppliers is to use a Universal Description, Discovery and Integration (UDDI) registry to search for potential suppliers. When the suppliers are chosen, the registry sends back information on how to contact the flower distributors that meet the criteria of the flower shop owner.

The flower shop owner can request price lists from each of the suppliers by obtaining a WSDL file for each potential supplier. The WSDL can be downloaded from the Web page of the supplier, received through e-mail, or retrieved from the UDDI registry entry of the supplier.

The WSDL describes the procedure call. When using the application server, the procedure call is a JAX-RPC or a JAX-WS procedure call. Either of these procedure call types retrieves the price list. The WSDL file also specifies the Universal Resource Locator (URL), where the request is sent.

The flower shop owner now has to compare the prices received from each supplier, decide which suppliers to do business with, and make arrangements for future orders to fill. The flower shop can now sell merchandise through the Web by using Web services to communicate with suppliers for the best prices and complete the ordering processes. The merchandise price lists need publishing to the Web site and a mechanism is needed for customers to order flowers.

The Web services clients of the flower supplier are deployed on the flower shop server. When a customer makes a transaction to purchase flowers through the Web, the order is sent to the supplier through the procedure call. The supplier responds by sending a confirmation with the order number and shipping date. The suppliers maintain the inventory and the flower shop owner handles billing and customer order management.

Similarly, the flower shop catalog can be composed automatically from the catalogs of each supplier. If the supplier delivers directly to the customer, then the order tracking inquiries can pass directly to the order tracking system of the supplier. The supplier can also use Web services to send invoices for orders by the

flower shop. Processes that previously required forms to fill manually, and fax or mail, can now be done automatically, saving labor costs for both the flower shop and the supplier.

Using Web services is beneficial because a much larger inventory is made available to the flower shop. No merchandise maintenance overhead exists, and the flower shop can offer their customers products that they otherwise might not have. Selling flowers through the Web increases capital for the flower shop without overhead of another store or resources invested into additional products.

For a more detailed scenario, see Web services scenario: Overview which tells the story of a fictional online garden supply retailer, Plants by WebSphere, and how they incorporated the Web services concept.

Refer to the Samples Gallery for additional Samples that demonstrate JAX-WS and JAX-RPC Web services.

Service-oriented architecture

A *service-oriented architecture* (SOA) is a collection of services that communicate with each other, for example, passing data from one service to another or coordinating an activity between one or more services.

Companies want to integrate existing systems to implement Information Technology (IT) support for business processes that cover the entire business value chain. A variety of designs are used, ranging from rigid point-to-point electronic data interchange (EDI) to Web auctions. By using the Internet, companies can make their IT systems available to internal departments or external customers, but the interactions are not flexible and are without standardized architecture.

Because of this increasing demand for technologies that support connecting and sharing resources and data, a need exists for a flexible, standardized architecture. SOA is a flexible architecture that unifies business processes by structuring large applications into building blocks, or small modular functional units or services, for different groups of people to use inside and outside the company. The building blocks can be one of three roles: service provider, service broker, or service requestor. See Web services approach to a service-oriented architecture to learn more about these roles.

Requirements for an SOA

To efficiently use an SOA, follow these requirements:

- **Interoperability between different systems and programming languages**.

  The most important basis for a simple integration between applications on different platforms is to provide a communication protocol. This protocol is available for most systems and programming languages.
- **Clear and unambiguous description language**.

  To use a service offered by a provider, it is not only necessary to be able to access the provider system, but the syntax of the service interface must also be clearly defined in a platform-independent fashion.
- **Retrieval of the service**.

  To support a convenient integration at design time or even system run time, a search mechanism is required to retrieve suitable services. Classify these services as *computer-accessible*, *hierarchical* or *taxonomies* based on what the services in each category do and how they can be invoked.

Web services approach to a service-oriented architecture

You can use Web services in a service-oriented architecture (SOA) environment.

You can use Web services to implement a SOA. A major focus of Web services is to make functional building blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled. A service can rely on another service to achieve its goals.

Each SOA building block can assume one or more of three roles:

- **Service provider**

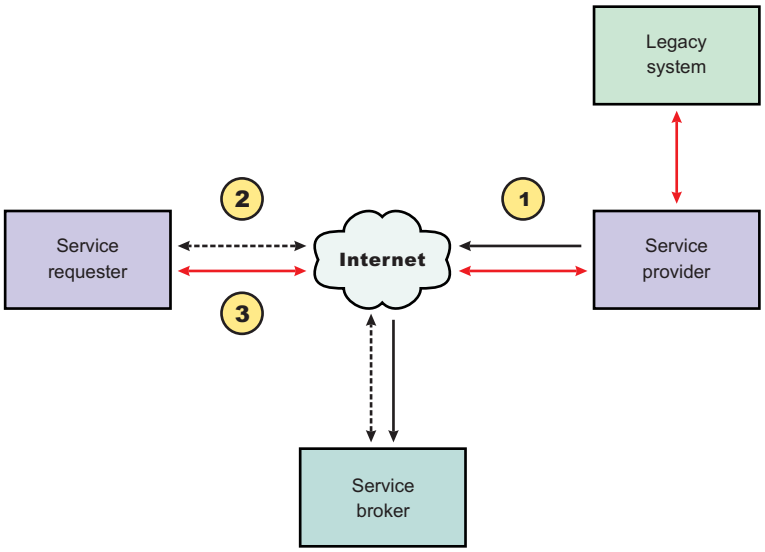
  The service provider creates a Web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services, or how to exploit free services for other value. The provider also has to decide which category to list the service in for a given broker service and what sort of trading partner agreements are required to use the service.

- **Service broker**

  The service broker, also known as *service registry*, is responsible for making the Web service interface and implementation access information available to any potential service requestor. The implementer of the broker decides the scope of the broker. Public brokers are available through the Internet, while private brokers are only accessible to a limited audience, for example, users of a company intranet. Furthermore, some decisions need to be made about the amount of the offered information. Some brokers specialize in many listings. Others offer high levels of trust in the listed services. Some cover a broad landscape of services and others focus within an industry. Some brokers catalog other brokers. Depending on the business model, brokers can attempt to maximize look-up requests, the number of listings or the accuracy of the listings. The Universal Description, Discovery and Integration (UDDI) specification defines a way to publish and discover information about Web services.

- **Service requester**

  The service requestor or Web service client locates entries in the broker registry using various find operations and then binds to the service provider to invoke one of its Web services.



Characteristics of the SOA

The presented SOA illustrates a loose coupling between the participants, which provides greater flexibility in the following ways:

- A client is coupled to a service. Therefore, the integration of the server takes place outside the scope of the client application programs.
- Old and new functional blocks or applications and systems, are encapsulated into components that work as services.
- Functional components and their interfaces are separate so that new interfaces can be plugged in more easily.
- Within complex applications, the control of business processes can be isolated. A business rule engine can be incorporated to control the workflow of a defined business process. Depending on the state of the workflow, the engine calls the respective services.

- Services can be incorporated dynamically during run time.
- Bindings are specified using configuration files and can be easily adapted to new needs.

Properties of a service-oriented architecture

The service-oriented architecture offers the following properties:

- **Web services are self-contained**

  On the client side, no additional software is required. A programming language with Extensible Markup Language (XML) and HTTP client support is enough to get you started. On the server side, a Web server and a SOAP server are required. It is possible to enable an existing application for Web services without writing a single line of code.
- **Web services are self-describing**

  Neither the client nor the server knows or cares about anything besides the format and content of the request and response messages (loosely coupled application integration). The definition of the message format travels with the message; no external metadata repositories or code generation tool are required.
- **Web services can be published, located, and invoked across the Internet**

  This technology uses established lightweight Internet standards such as HTTP and it leverages the existing infrastructure. Some other standards that are required include, SOAP, Web Services Description Language (WSDL), and UDDI.
- **Web services are language-independent and interoperable**

  The client and server can be implemented in different environments. Existing code does not have to change in order to be Web services-enabled.
- **Web services are inherently open and standard-based**

  XML and HTTP are the major technical foundation for Web services. A large part of the Web service technology has been built using open-source projects.
- **Web services are dynamic**

  Dynamic e-business can become reality using Web services because with UDDI and WSDL you can automate the Web service description and discovery.
- **Web services are composable**

  Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation. Web services can be chained together to perform higher-level business functions. This chaining shortens development time and enables best-of-breed implementations.
- **Web services are loosely coupled**

  Traditionally, application design has depended on tight interconnections at both ends. Web services require a simpler level of coordination that supports a more flexible reconfiguration for an integration of the services.
- **Web services provide programmatic access**

  The approach provides no graphical user interface; it operates at the code level. Service consumers need to know the interfaces to Web services, but do not need to know the implementation details of services.
- **Web services provide the ability to wrap existing applications**

  Already existing stand-alone applications can easily integrate into the SOA by implementing a Web service as an interface.

Web services business models supported

This article explains the concept and business models that can be implemented by using Web services in a service-oriented architecture (SOA).

The properties and benefits of using a SOA such as Web services is well suited for binding small modules that perform independent tasks within a highly heterogeneous e-business model. Web services can be easily wrapped around existing applications in your business model and plugged into different business processes.

For connecting to a large monolithic system that does not support the implementation of different flexible business processes, other approaches might be better suited, for example, to satisfy specialized features, such as performance or security.

The following business models are easily implemented by using an architecture including Web services:

- **Business information**

  Sharing of information with consumers or other businesses. Web services can be used to expand the reach through such services as news streams, local weather reports, integrated travel planning, and intelligent agents.
- **Business integration**

  Providing transactional, fee-based services for customers. A global network of suppliers can be easily created. Web services can be implemented in auctions, e-marketplaces, and reservation systems.
- **Business process externalization**

  Web services can be used to model value chains by dynamically integrating processes to a new solution within an organizational unit or even with those of other e-businesses. This modeling can be achieved by dynamically linking internal applications to new partners and suppliers, to offer their services to complement internal services.

To see how these models are implemented using all aspects of Web services, see Web services scenario: Overview which tells the story of a fictional online garden supply retailer named Plants by WebSphere and how this retailer incorporates the Web services concept.

Web services

Web services are self-contained, modular applications that you can describe, publish, locate, and invoke over a network.

The application server supports Web services that are developed and implemented based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification. The application server supports the Java API for XML Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. The JAX-WS is a strategic programming model that simplifies application development through support of a standard, annotation-based model to develop Web services applications and clients.

A typical Web services scenario is a business application requesting a service from another existing application. The request is processed through a given Web address using SOAP messages over a HTTP, Java Message Service (JMS) transport or invoked directly as Enterprise JavaBeans (EJB). The service receives the request, processes it, and returns a response. Examples of a simple Web service include weather reports or getting stock quotes. The method call is synchronous, that is, the method waits until the result is available. Transaction Web services, supporting quotes, business-to-business (B2B) or business-to-client (B2C) operations include airline reservations and purchase orders.

Web services can include the actual service or the client that accesses the service.

Web services are Web applications that help improve the flexibility of your business processes by integrating with applications that otherwise do not communicate. The inner-library loan program at your local library is a good example of the Web services concept and its evolution. The Web service concept existed even before the term; the concept became widely accepted with the creation of the Internet. Before the Internet was created, you visited your library, searched the collections and checked out your books. If you did not find the book that you wanted, the librarian ran a search for you by computer or phone and

located the book at a nearby library. The librarian ordered the book for you and you picked it up after it was delivered to your local library. By incorporating Web services applications, you can streamline your library visit.

Now, you can search the local library collection and other local libraries at the same time. When other libraries provide your library with a Web service to search their collection (the service might have been provided through Universal Description Discovery and Integration (UDDI), your results yield their resources. You might use another Web service application to check out and send the book to your home. Using Web services applications saves time and provides a convenience for you, as well as freeing the librarian to do other business tasks.

Web services reflect the service-oriented architecture (SOA) approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, Web services applications provide components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

The key components of Web services include:

- Web Services Description Language (WSDL)

  WSDL is the XML-based file that describes the Web service. The Web service request uses this file to bind to the service.
- SOAP

  SOAP is the XML-based protocol that the Web service request uses to invoke the service.

For a more detailed scenario, see Web services scenario: Overview, which tells the story of a fictional online garden supply retailer named Plants by WebSphere, and how this retailer incorporated the Web services concept.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Web Services for Java EE specification

The *Web Services for Java Platform, Enterprise Edition (Java EE)* specification defines the programming model and runtime architecture for implementing Web services based on the Java language. Another name for the Web Services for Java EE specification is the Java Specification Requirements (JSR) 109. The specification includes open standards for developing and implementing Web services.

The Web Services for Java EE specification is based on the Java EE technology and supports the Java API for XML Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) programming model for Web services and clients in a manner that is interoperable and portable across application servers within environments that are scalable and secure. This specification is based on industry standards for Web services, including Web Services Description Language (WSDL) and SOAP, and it describes the development and deployment of Web services.

The application server supports the Web Services for Java EE specification, Version 1.2. This specification supports WSDL Version 1.1, SOAP Version 1.1 and SOAP Version 1.2. Prior to Web Services for Java EE Version 1.2, the JSR 109 specification name was Web Services for Java 2 Platform, Enterprise Edition (J2EE).

You can integrate the Java EE technology with Web services in a variety of ways. You can expose Java EE components as Web services, for example, JavaBeans and enterprise beans. When you expose Java EE components as Web services, clients that are written in Java code or existing Web service clients that are not written in Java code can access these services. Java EE components can also act as Web service clients.

The Web Services for Java EE specification is the preferred platform for Web-based programming because it provides open standards permitting different types of languages, operating systems and software to communicate seamlessly through the Internet.

For a Java application to act as Web service client, a mapping between the WSDL file and the Java application must exist. For JAX-WS applications, the mapping is defined using annotations. You can optionally use the webservices.xml deployment descriptor to specify the location of the WSDL file and override the value defined in the @WebServices annotation. For JAX-RPC applications, you must define the JAX-RPC mapping file. To learn more about the mapping that is defined between the WSDL file and your Web service application, see the JAX-WS specification or the JAX-RPC specification in the Web services specifications and API documentation depending on the programming model used.

You can use a Java component to implement a Web service by specifying the component interface and binding information in the WSDL file and designing the application server infrastructure to accept the service request.

This entire process encompassed is based on the Web Services for Java EE specification.

The specification defines the webservices.xml deployment descriptor specifically for Web services. The webservices.xml deployment descriptor file defines the set of Web services that you can deploy in a Web Services for Java EE enabled container.

For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

For example, if your service implementation class for your JAX-WS Web service includes the @WebService annotation as follows:

```
@WebService(wsdlLocation="http://myhost.com/location/of/the/wsdl/ExampleService.wsdl")
```

and the webservices.xml specifies a different filename for the WSDL document as follows:

```
<webservices>
<webservice-description>
<webservice-description-name>ExampleService</webservice-description-name>
<wsdl-file>META-INF/wsdl/ExampleService.wsdl</wsdl-file>
...
</webservice-description>
</webservices>
```

then the value that is specified in the deployment descriptor, META-INF/wsdl/ExampleService.wsdl overrides the annotation value.

See section 5 of the Web Services for Java EE specification for details regarding the correlation between values specified in the Web services deployment descriptor file and the attributes of the @WebService and the @WebServiceProvider annotations.

For JAX-RPC Web services, you must define the deployment characteristics in the webservices.xml deployment descriptor file.

You are responsible for providing various elements to the deployment descriptor, including:

- Port name
- Port service implementation
- Port service endpoint interface
- Port WSDL definition

- Port QName
- MTOM/XOP support for JAX-WS Web services
- Protocol binding for JAX-WS Web services
- JAX-RPC mapping
- Handlers (optional)
- Servlet mapping (optional)

The Enterprise JavaBeans (EJB) 2.1 specification also states that for a Web service developed from a session bean, the EJB deployment descriptor, `ejb-jar.xml`, must contain the service-endpoint element. The service-endpoint value must be the same as that stated in the webservices.xml deployment descriptor.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Artifacts used to develop Web services

With *development artifacts* you can develop an enterprise bean or a JavaBeans module into a Web service. This topic describes artifacts used to develop Web services that are based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification.

To create a Web service from an enterprise bean or from a JavaBeans module, the following files are added to the respective Java archive (JAR) file or Web archive (WAR) modules at assembly time:

- **Web Services Description Language (WSDL) Extensible Markup Language (XML) file**

  The WSDL XML file describes the Web service that is implemented.
- **Service Endpoint Interface**

  A Service Endpoint Interface is the Java interface corresponding to the Web service port type implemented. The Service Endpoint Interface is defined by the Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) Web services run time that you are using.
- **webservices.xml**

  The webservices.xml file contains the Java EE deployment descriptor of the Web service specifying how the Web service is implemented. The webservices.xml file is defined in the Web Services for Java EE specification.

  For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

  For JAX-RPC applications, deployment descriptors are required to specify how the Web service is implemented.
- **ibm-webservices-bnd.xmi (JAX-RPC applications only)**

  This file contains WebSphere product-specific deployment information and is defined in ibm-webservices-bnd.xmi assembly properties.
- **Java API for XML-based remote procedure call (JAX-RPC) mapping file**

  The JAX-RPC mapping deployment descriptor specifies how Java elements are mapped to and from WSDL file elements.

The following files are added to an application client, enterprise beans or Web module to permit a Web Services for Java Platform, Enterprise Edition (Java EE) client access to Web services:

- **WSDL file**

  The WSDL file is provided by the Web service implementer.
- **Java interfaces for the Web service**

  The Java interfaces are generated from the WSDL file as specified by the JAX-WS or JAX-RPC specification. These bindings are the Service Endpoint Interface based on the WSDL port type, or the service interface, which is based on the WSDL service.

- **ibm-webservicesclient-bnd.xmi (JAX-RPC applications only)**

  This file contains WebSphere product-specific deployment information, such as security information for JAX-RPC applications. For JAX-WS applications, deployment descriptors are not supported and have been replaced by the use of annotations.
- **Other JAX-RPC binding files**

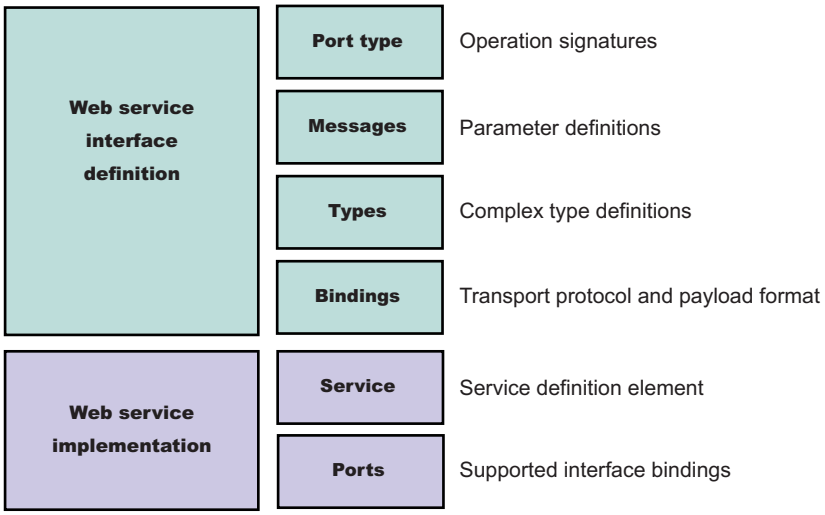
  Additional JAX-RPC binding files that support the client application in mapping SOAP to the Java language are generated from WSDL by the WSDL2Java command tool.

WSDL

Web Services Description Language (WSDL) is an Extensible Markup Language (XML)-based description language. This language was submitted to the World-Wide Web Consortium (W3C) as the industry standard for describing Web services. The power of WSDL is derived from two main architectural principles: the ability to describe a set of business operations and the ability to separate the description into two basic units. These units are a description of the operations and the details of how the operation and the information associated with it are packaged.

A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definitions of endpoints and messages are separated from their concrete network deployment or data format bindings. This separation supports the reuse of abstract definitions: messages, which are abstract descriptions of exchanged data, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports defines a service. Therefore, a WSDL document is composed of several elements.

The following is the structure of the information in a WSDL file:



A WSDL file contains the following parts:

- **Web service interface definition**

  This part contains the elements and the namespaces.
- **Web service implementation**

  This part contains the definition of the service and ports.

A WSDL file describes a Web service with the following elements:

portType

The description of the operations and associated messages. The portType element defines abstract operations.

```
<portType name="EightBall">
 <operation name="getAnswer">
  <input message="ebs:IngetAnswerRequest"/>
  <output message="ebs:OutgetAnswerResponse"/>
 </operation>
</portType>
```

message

The description of input and output parameters and return values.

```
<message name="IngetAnswerRequest">
 <part name="meth1_inType" type="ebs:questionType"/>
</message>
<message name="OutgetAnswerResponse">
 <part name="meth1_outType" type="ebs:answerType"/>
</message>
```

types

The schema for describing XML types used in the messages.

```
<types>
 <xsd:schema targetNamespace="...">
  <xsd:complexType name="questionType">
   <xsd:element name="question" type="string"/>
  </xsd:complexType>
  <xsd:complexType name="answerType">
  ...
</types>
```

binding

The bindings describe the protocol that is used to access a portType, as well as the data formats for the messages that are defined by a particular portType element.

```
<binding name="EightBallBinding" type="ebs:EightBall">
 <soap:binding style="rpc" transport="schemas.xmlsoap.org/soap/http">
 <operation name="ebs:getAnswer">
 <soap:operation soapAction="urn:EightBall"/>
  <input>
   <soap:body namespace="urn:EightBall" ... />
  ...
```

Service

The services and ports define the location of the Web service.

The service contains the Web service name and a list of ports.

Ports

The ports contain the location of the Web service and the binding used for service access.

```
<service name="EightBall">
 <port binding="ebs:EightBallBinding" name="EightBallPort">
  <soap:address location="localhost:8080/axis/EightBall"/>
 </port>
</service>
```

When creating Java API for XML Web Services (JAX-WS) or Java API for XML-based RPC (JAX-RPC) Web services, you can use a bottom-up development approach when you start from JavaBeans or an enterprise bean, or you can use a top-down development approach when you start with an existing Web Services Description Language (WSDL) file.

When creating JAX-WS Web services for this product, you can start with either a WSDL or an implementation bean class. If you start with an implementation bean class, then use the wsgen command line tool to generate all the Web services server artifacts, including a WSDL if requested. If you start with a WSDL, then use the wsimport command line tool to generate all the Web services artifacts for either the server or client side.

When creating JAX-RPC Web services for this product, you must first have an implementation bean that includes a service endpoint interface. Then, you use the Java2WSDL command-line tool to create a WSDL file that defines the Web services. If you are starting with the WSDL to generate the implementation bean class, run the WSDL2Java command line tool against the WSDL file to create Java APIs and deployment descriptor templates.

Multipart WSDL and WSDL publication

The product supports deployment of Web services using a multipart Web Services Description Language (WSDL) file. In multipart WSDL files, an implementation WSDL file contains the wsdl:service. This implementation WSDL file imports an interface WSDL file, which contains the other WSDL constructs. This supports multiple Web services using the same WSDL interface definition.

The <wsdl:import> element indicates a reference to another WSDL file. If the <wsdl:import> element location attribute does not contain a URL, that is, it contains only a file name, and does not begin with http://, https:// or file://, the imported file must be located in the same directory and must not contain a relative path component. For example, if META-INF/wsdl/A_Impl.wsdl is in your module and contains the <wsdl:import=″A.wsdl″ namespace=″...″/> import statement, the A.wsdl file must also be located in the module META-INF/wsdl directory.

It is recommended that you place all WSDL files in either the META-INF/wsdl directory, if you are using Enterprise JavaBeans (EJB), or the WEB-INF/wsdl directory, if you are using JavaBeans components, even if relative imports are located within the WSDL files. Otherwise, implications exist with the WSDL publication when you use a path like <location=″../interfaces/A_Interface.wsdl″namespace=″...″/>. Using a path like this example fails because the presence of the relative path, regardless of whether the file is located at that path or not. If the location is a Web address, it must be readable at both deployment and server startup.

You can publish the files located in the META-INF/wsdl or the WEB-INF/wsdl directory through either a URL address or file, including WSDL or XML Schema Definition (XSD) files. For example, if the file referenced in the <wsdl-file> element of the webservices.xml deployment descriptor is located in the META-INF/wsdl or the WEB-INF/wsdl directory, it is publishable. If the files imported by the <wsdl-file> are located in the wsdl/ directory or its subdirectory, they are publishable.

If the WSDL file referenced by the <wsdl-file> element is located in a directory other than wsdl, or its subdirectories, the file and its imported files, either WSDL or XSD files, which are in the same directory, are copied to the wsdl directory without modification when the application is installed. These types of files can also be published.

If the <wsdl-file> imports a file located in a different directory (a directory that is not -INF/wsdl or a subdirectory), the file is not copied to the wsdl directory and not available for publishing.

For JAX-WS Web services, you can use an annotation to specify the location of the WSDL. Use the @WebService annotation with the WSDLLocation attribute. The WSDLLocation attribute is optional. If this attribute is not specified, then WSDL is generated and published from the information that is found in the

Web service classes. You can optionally specify the location of the WSDL file in the webservices.xml deployment descriptor. However, any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

SOAP

SOAP is a specification for the exchange of structured information in a decentralized, distributed environment. As such, it represents the main way of communication between the three key actors in a service oriented architecture (SOA): service provider, service requestor and service broker. The main goal of its design is to be simple and extensible. A SOAP message is used to request a Web service.

SOAP 1.1

WebSphere Application Server follows the standards outlined in SOAP 1.1.

SOAP was submitted to the World Wide Web Consortium (W3C) as the basis of the Extensible Markup Language (XML) Protocol Working Group by several companies, including IBM and Lotus. This protocol consists of three parts:

- An *envelope* that defines a framework for describing message content and processing instructions.
- A set of *encoding rules* for expressing instances of application-defined data types.
- A *convention* for representing remote procedure calls and responses.

SOAP 1.1 is a protocol-independent transport and can be used in combination with a variety of protocols. In Web services that are developed and implemented with WebSphere Application Server, SOAP is used in combination with HTTP, HTTP extension framework, and Java Message Service (JMS). SOAP is also operating-system independent and not tied to any programming language or component technology.

As long as the client can issue XML messages, it does not matter what technology is used to implement the client. Similarly, the service can be implemented in any language, as long as the service can process SOAP messages. Also, both server and client sides can reside on any suitable platform.

SOAP 1.2

The SOAP 1.2 specification is also a W3C recommendation, and WebSphere Application Server follows the standards that are outlined in SOAP 1.2. The SOAP 1.2 specification comes in three parts plus some assertions and a test collection:

- Part 0: Primer
- Part 1: Messaging Framework
- Part 2: Adjuncts
- Specification Assertions and Test Collection

SOAP 1.2 provides a more specific definition of the SOAP processing model, which removes many of the ambiguities that sometimes led to interoperability problems in the absence of the Web Services-Interoperability (WS-I) profiles. SOAP 1.2 should reduce the chances of interoperability issues with SOAP 1.2 implementations between different vendors.

Some of the more significant changes in the SOAP 1.2 specification include:

- The ability to now officially define other transport protocols other than the HTTP protocol as long as vendors conform to the binding framework that is defined in SOAP 1.2. While HTTP is ubiquitous, it is not as reliable of a transport as other things such as TCP/IP, MQ, and so forth.
- The fact that SOAP 1.2 is based on the XML Information Set (XML Infoset). The information set provides a way to describe the XML document using the XSD schema but does not necessarily serialize the document by using XML 1.0 serialization. SOAP 1.1 is based upon XML 1.0 serialization. The information set will make it easier to use other serialization formats such as a binary protocol format.

You can use a binary protocol format shrink the message into a much more compact format where some of the verbose tagging information might not be required.

The Java API for XML Web Services (JAX-WS) standard introduces the ability to support both SOAP 1.1 as well as SOAP 1.2.

See “Differences in SOAP versions” on page 352 for additional differences between SOAP 1.1 and SOAP 1.2.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

SOAP with Attachments API for Java interface:

The *SOAP with Attachments API for Java* (SAAJ) interface is used for SOAP messaging that provides a standard way to send XML documents over the Internet from a Java programming model. SAAJ is used to manipulate the SOAP message to the appropriate context as it traverses through the runtime environment.

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

The Java API for XML-Based RPC (JAX-RPC) programming model supports SAAJ 1.2 to manipulate the XML.

The JAX-WS programming model supports SAAJ 1.2 and 1.3. SAAJ 1.3 includes support for SOAP 1.2 messages.

The differences in the SAAJ 1.2 and SAAJ 1.3 specification can be reviewed in the topic ″Differences in SAAJ versions.″

How are messages used in Web services?

Web services use XML technology to exchange messages. These messages conform to XML schema. When developing Web services applications, there are limited XML APIs to work with, for example, Document Object Model (DOM). It is more efficient to manipulate the Java objects and have the serialization and deserialization completed during run time.

Web services uses SOAP messages to represent remote procedure calls between the client and the server. Typically, the SOAP message is deserialized into a series of Java value-type business objects that represent the parameters and return values. In addition, the Java programming model provides APIs that support applications and handlers to manipulate the SOAP message directly. Because there are a limited number of XML schema types that are supported by the programming models, the specification provides the SAAJ data model as an extension to manipulate the message.

To manipulate the XML schema types, you need to map the XML schema types to Java types with a *custom data binder*.

The SAAJ interface

The SAAJ-related classes are located in the javax.xml.soap package. SAAJ builds on the interfaces and abstract classes and many of the classes begin by invoking factory methods to create a factory such as SOAPConnectionFactory and SOAPFactory.

The most commonly used classes are:

- SOAPMessage: Contains the message, both the XML and non-XML parts
- SOAPHeader: Represents the SOAP header XML element
- SOAPBody: Represents the SOAP body XML element
- SOAPElement: Represents the other elements in the SOAP message

Other parts of the SAAJ interface include:

- MessageContext: Contains a SOAP message and related properties
- AttachmentPart: Represents a binary attachment
- SOAPPart: Represents the XML part of the message
- SOAPEnvelope: Represents the SOAP envelope XML element
- SOAPFault: Represents the SOAP fault XML element

The primary interface in the SAAJ model is javax.xml.soap.SOAPElement, also referred to as SOAPElement. Using this model, applications can process an SAAJ model that uses pre-existing DOM code. It is also easier to convert pre-existing DOM objects to SAAJ objects.

Messages created using the SAAJ interface follow SOAP standards. A SOAP message is represented in the SAAJ model as a javax.xml.soap.SOAPMessage object. The XML content of the message is represented by a javax.xml.soap.SOAPPart object. Each SOAP part has a SOAP envelope. This envelope is represented by the SAAJ javax.xml.SOAPEnvelope object. The SOAP specification defines various elements that reside in the SOAP envelope; SAAJ defines objects for the various elements in the SOAP envelope.

The SOAP message can also contain non-XML data that is called attachments. These attachments are represented by SAAJ AttachmentPart objects that are accessible from the SOAPMessage object.

A number of reasons exist as to why handlers and applications use the generic SOAPElement API instead of a tightly bound mapping:

- The Web service might be a conduit to another Web service. In this case, the SOAP message is only forwarded.
- The Web service might manipulate the message using a different data model, for example a Service Data Object (SDO). It is easier to convert the message from a SAAJ DOM to a different data model.
- A handler, for example, a digital signature validation handler, might want to manipulate the message generically.

You might need to go a step further to map your XML schema types, because the SOAPElement interface is not always the best alternative for legacy systems. In this case you might want to use a generic programming model, such as SDO, which is more appropriate for data-centric applications.

The XML schema can be configured to include a custom data binding that pairs the SDO or data object with the Java object. For example, the run time renders an incoming SOAP message into a SOAPElement interface and passes it to the customer data binder for more processing. If the incoming message contains an SDO, the run time recognizes the data object code, queries its type mapping to locate a custom binder, and builds the SOAPElement interface that represents the SDO code. The SOAPElement is passed to the SDOCustomBinder.

See Custom data binders for more information about the process of developing applications with SOAPElement, SDO and custom binders.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Related concepts

“SOAP” on page 341
SOAP is a specification for the exchange of structured information in a decentralized, distributed environment. As such, it represents the main way of communication between the three key actors in a service oriented architecture (SOA): service provider, service requestor and service broker. The main goal of its design is to be simple and extensible. A SOAP message is used to request a Web service.

“Differences in SOAP versions” on page 352
Both SOAP Version 1.1 and SOAP Version 1.2 are World Wide Web Consortium (W3C) standards. Web services can be deployed that support not only SOAP 1.1 but also support SOAP 1.2. Some changes from SOAP 1.1 that were made to the SOAP 1.2 specification are significant, while other changes are minor.

“Differences in SAAJ versions”
The SOAP with Attachments API for Java (SAAJ) interface Version 1.3 expands the support of SOAP 1.2 messages in a Web services environment. There are several differences between SAAJ 1.2 and SAAJ 1.3 that are presented in this topic.

Related reference

Web services specifications and APIs

Differences in SAAJ versions:

The SOAP with Attachments API for Java (SAAJ) interface Version 1.3 expands the support of SOAP 1.2 messages in a Web services environment. There are several differences between SAAJ 1.2 and SAAJ 1.3 that are presented in this topic.

In a typical Web services environment, you rely on the underlying code that is based on Java standards to translate a set of Java objects. The SAAJ interface provides APIs to read, write, send and receive SOAP messages, and advocates binary content sent as an attachment to a SOAP message.

SAAJ 1.3 aligns with SOAP 1.1 and SOAP 1.2 messages and is supported by the Java API for XML Web Services (JAX-WS) programming model and the Java API for XML-Based RPC (JAX-RPC) programming model. SAAJ 1.2 works with SOAP 1.1 messages only.

If you migrate your code from SOAP 1.1 to SOAP 1.2, you can continue to use your existing SOAP 1.1 code, if the message is a SOAP 1.2 message. If you upgrade your base code to use SAAJ 1.3, then you can continue to use the existing code that operates on a SOAP 1.1 message. An example of these differences is in SOAP 1.1, where the human readable text of a fault is stored in the faultString element. In SOAP 1.2, the human readable text is stored in the Reason element. Your code might look like the following example:

```
String text = soapFault.getFaultString();
```

The getFaultString () returns the faultString value if the message is based on SOAP 1.1. If you are using SOAP 1.2, the getFaultString () returns the Reason value. In addition, the SAAJ 1.3 interface provides a new method, getReasonText (Locale), that gets a specific Reason value. The getReasonText (Locale) method returns a documented exception if the message is based on SOAP 1.1. The SAAJ 1.3 interface supports existing code to process both SOAP 1.1 and SOAP 1.2 messages.

Other differences between SAAJ 1.2 and SAAJ 1.3 are in the following list:

- SAAJMetaFactory interface

  The SAAJMetaFactory SPI is introduced to support creating SOAP factory classes in a single place.
- SAAJResult class

The SAAJResult object acts as a holder for the results of a Java API for XML Processing (JAX-P) transformation or a Java Architecture for XML Binding (JAXB) marshalling, in the SAAJ tree. The SAAJResult class is introduced for improved usability when transformation results are expected to be a valid SAAJ tree.

- Overloaded methods that accept a QName instead of a Name

  It is preferred that a QName represents an XML-qualified name. Therefore, overloaded methods are introduced in all of the SAAJ APIs, where a corresponding method accepts a javax.xml.soap.Name name as an argument.
- New methods in AttachmentPart, SOAPBody and SOAPElement interfaces and classes

  Use these new methods to assist you when you are working with the new SOAP features.
- SOAPPart is now a javax.xml.soap.Node method.

  The SOAPPart object is now also considered to be a SOAP node method.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Message Transmission Optimization Mechanism:

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

There are many reasons why you might want to send binary attachments, such as images or files, along with a Web services request. There are ways to accomplish this, such as:

- Encoding with base64 inline in the SOAP payload. However, encoding inline tends to enlarge the size of the SOAP message. Note that base64 encoding might double the size of the binary data.
- Encoding the messages by using SOAP with Attachments (SwA) and to follow the Web Services Interoperability Organization (WS-I) Attachments Profile. WebSphere Application Server currently supports this method.
- Providing optimization of binary message transportation by using XML-binary Optimized Packaging (XOP). Optimization is available only for binary data or content. MTOM uses XOP in the context of SOAP and MIME over HTTP.

XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that just happens to look like a MIME multipart/related package, with an XML documents as the root part. That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that is associated with encoding. Encoding is the only way a binary data can fit directly into an XML world.

If MTOM mapping generation is disabled, then XOP is disabled. If XOP is disabled, the binary data are not sent by using MIME attachments. Instead, the binary data is base64 encoded as usual.

The MTOM specification is defined in three different parts:

- An abstract feature for optimized transmission or wire format for SOAP messages. This feature is abstract in the sense that the description of the optimization technique as well as the behavior of the SOAP processors at sender, receiver and intermediaries is generic and does not include any references to technologies such as MIME, HTTP, and so forth. The optimization technique centers around ensuring a SOAP envelope Infoset view for the SOAP processors while encoding selectively certain contents of the SOAP Envelope Infoset that are accessible as canonical lexical representation of the xs:base64Binary data type.

Implementing these abstract features requires concrete specification of two aspects: the optimized wire format and how the optimized wire format flows on a particular transport

- The second part of the MTOM specification addresses the serialization aspect and depends normatively upon MIME Multipart/Related XOP packaging. The serialization aspect is where MTOM relates to XOP.
- As a concrete SOAP HTTP binding level feature, MTOM expands upon the serialization. This part describes how HTTP binding can be used to transport the XOP packages that are holding the SOAP MTOM messages. This part also puts some restrictions on the possible serializations of the SOAP MTOM messages as XOP packages, such as use of XML 1.0 only.

The Java API for XML Web Services (JAX-WS) adds support for sending binary data attachments using MTOM. JAX-WS is the centerpiece of a newly re-architected API stack for Web service that includes JAX-WS 2.0, JAXB 2.0, and SAAJ 1.3. The API stack is sometimes referred to as the integrated stack. JAX-WS is designed to take the place of JAX-RPC in Web services and Web applications.

Attachment approach

Attachment by value or by reference has been the widely accepted technique for handling opaque data in XML-formatted messages.

- **By value** is when the opaque data content is embedded as an element or as an attribute by using either base64 or hexadecimal text encoding approach, which is codified in the XML schema as data types xs:base64Binary and xs:hexBinary, respectively.
- **By reference** is when the opaque data content is referenced externally as element or as attribute by using a URI, which is codified in the XML schema as data type xs:anyURI.

The use of either of these two techniques has its advantages and disadvantages. MTOM is the specification that is focused on resolving these inherent attachments problems.

A different standard is defined by World Wide Web Consortium (W3C) and is called SOAP with Attachments (SwA). SwA was developed as a way to package SOAP messages with attachments. Because some vendors do not support SwA, SwA can be replaced by the more powerful MTOM and XOP mechanisms. SwA and MTOM are conceptually similar, and both encode binary data as a MIME attachment in a MIME document. Using MIME attachments improves the performance of large binary payloads transport.

Additional differences between SwA and MTOM include:

- MTOM uses a standard called XOP, which defines a XOP reference that exists within the SOAP payload. This reference points to the MIME attachment that contains the binary data.
- With MTOM, the XOP reference logically includes the binary data into the XML Information Set (Infoset). With SwA, the href points to data that is not only physically outside the XML document but is not logically included within its Infoset.
- With MTOM, binary attachments can be logically signed as if they were part of the SOAP XML document.
- In additional to IBM, Microsoft® .NET supports MTOM, which eliminates some of the interoperability problems found with SwA. Interoperability was treated as the main goal when the co-submitters discussed the suggested modifications.

The MTOM attachment approach takes advantage of the SOAP infrastructure while also gaining transport efficiencies that are provided by SOAP with Attachment (SwA) solution.

SOAP 1.2 and SOAP 1.1

SOAP 1.1 is based on the XML specification. Likely, the SOAP 1.1 implementation will continue to exist for a few years. For those who are still running SOAP 1.1, there is now an interoperable way to use MTOM for attachments support. SAP, Oracle, Microsoft, and IBM have submitted a SOAP 1.1 Binding for MTOM 1.0 specification to W3C, which defines how MTOM can be used with SOAP 1.1 payloads. The

specification details the necessary modifications to the SOAP MTOM and XOP specifications that are necessary to successfully use these technologies with SOAP 1.1. See the information at this Web site:http://schemas.xmlsoap.org/soap/mtom/SOAP11MTOM10.pdf

MTOM is a SOAP Version 1.2 feature, which is based on the Infoset. For more information, see “XML information set” on page 350.

Without MTOM, the data is encoded in whatever format is described in the schema (base64 or hex) and then is contained in the XML document. The following example shows a SOAP message with an <xsd:base64Binary> element:

```
... other transport headers ...
Content-Type: text/xml; charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
      <soapenv:Header/>
      <soapenv:Body>
         <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
            <input>
               <imageData>R0lGODl  ... more base64 encoded data ...  KTJk8giAAA7</imageData>
            </input>
         </sendImage>
      </soapenv:Body>
   </soapenv:Envelope>
```

When MTOM is enabled, the binary data that represents the attachment is included as a MIME attachment to the SOAP message. The following example shows an MTOM-enabled SOAP message with attachment data:

```
... other transport headers ...
Content-Type: multipart/related; boundary=MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812;
type="application/xop+xml"; start="<0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>";
start-info="text/xml"; charset=UTF-8

--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: application/xop+xml; charset=UTF-8; type="text/xml";
content-transfer-encoding: binary
content-id:
   <0.urn:uuid:0FE43E4D025F0BF3DC11582467646813@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
         <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
            <soapenv:Header/>
            <soapenv:Body>
               <sendImage xmlns="http://org/apache/axis2/jaxws/sample/mtom">
                  <input>
                     <imageData>
                        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
                             href="cid:1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org"/>
                     </imageData>
                  </input>
               </sendImage>
            </soapenv:Body>
         </soapenv:Envelope>
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812
content-type: text/plain
content-transfer-encoding: binary
content-id:
         <1.urn:uuid:0FE43E4D025F0BF3DC11582467646811@apache.org>

... binary data goes here ...
--MIMEBoundaryurn_uuid_0FE43E4D025F0BF3DC11582467646812--
```

XML-binary Optimized Packaging:

XML-binary Optimized Packaging (XOP) specification was standardized by the World Wide Web (W3C) on January 25, 2005. SOAP Message Transmission Optimization Mechanism (MTOM) uses XOP in the context of SOAP and MIME over HTTP.

XML is widely used for data transfer. XML is a popular format for exchanging well-formed documents because it is plain text, human-readable, and structured. For example, SOAP messaging in Web services is based on XML (or is based on XML Infoset with SOAP 1.2). People want to leverage legacy formats like PDF, GIF, JPEG and similar things, while still using an XML model. The desire to integrate XML with pre-existing data formats has been a long-standing and persistent issue for the XML community. Users often want to leverage the structured, extensible markup conventions of XML without abandoning existing data formats that do not readily adhere to XML 1.0 syntax.

As SOAP messaging in Web services becomes more widespread, the next step is how to send non-text based data, such as images and workflow data, along with your message. For example, you might want to send a scanned document in .jpeg format between two applications. The question becomes whether this data can be understood between the various applications.

Much of the value of XML and Web services resides in the ability to use generic XML tools to work with content. Many XML tools and standards for describing and manipulating XML (such as parsers, XPath, XQuery, XSLT, XML encryption and digital signature and XML schema) are not designed to work with non-text data, such as images. These XML tools do not work with non-XML content; these tools require text. The challenge is how non-text data (also called *binary data*) can be embedded or attached with XML. In other words, a way to attach a binary file to a SOAP message is needed.

Encoding is the only way binary data can fit directly into an XML world. Normally, you can embed binary data in an XML document by encoding it as text using Base 64. Base 64 is a serialization that has existed for some time, can be easily implemented out of the box, and has interoperability across platforms. The xsi:base64binary datatype supports this serialization in the XML Schema. Base 64 encodes your binary data into a textual representation that can squeeze into an XML document. Base 64 takes your binary data and translates it into a series of ASCII characters by encoding three octets at a time. Because each octet consists of eight bits, representing them as four printable characters in the ASCII standard, it uses 64 ASCII characters to represent the binary. All platforms can decode and encode using this convention. 6-bit ASCII is widely supported, and no special characters need to be dealt with. However, there is a performance impact for larger messages.

For applications that require speedy operation, Base 64 might not be the solution. If you want to index into such content, query it, transform it, encrypt it, sign it, or describe it, you need to use a different mechanism.

The first attachment specification known as SOAP with Attachments (SwA) was developed. The basic idea of SwA is that the binary message part (the attachment) is thought of as a Multipurpose Internet Mail Extensions (MIME) attachment. MIME is a widely implemented specification for formatting non-ASCII mail message attachments. SwA specifies that the SOAP body can contain a reference to the MIME message part (the attachment) simply by using a URI. The binary part is attached by a reference.

A few disadvantages of SwA include:

- SwA fails in its usability or interoperability. The SOAP infrastructure was created around the SOAP envelope, which didn't work well for attachments. An attachment using SwA means that two data models are used in one message. These two data models do not operate with existing XML technology.
- SwA does not work with the composable character of SOAP. Basically standards, such as WS-Security, were not written to work with attachments. WS-Security needs to work on all the data that needs to be digitally signed or encrypted, and that means all the data in the attachment also. But if it cannot access it, then it will not work and the signature is effectively invalid.

Often, users want to leave their existing non-XML formats as is, to be treated as opaque sequences of octets by XML tools and infrastructure. Such an approach permits widely used formats such as .jpeg and .wav to peacefully coexist with XML. XOP makes it a bit more realistic to use base64-encoded data. At the current time, XOP only permits base64-encoded data to be optimized.

Using XOP means that instances of XML-type base64Binary, if enabled, are transported by using MIME attachments. If XOP is in use, the implementation can automatically encode it. XOP maintains the data model of the XML message because the attachment is treated as base64-encoded data. If an XML stack understands XOP encoding, your application does not need to be changed at all. For example, when it wants to access a .jpeg picture, it can get the character value of the content as a base64-encoded string.

XOP gives people a way to think about MIME messages in a data exchange that they are comfortable with and already use for a lot of other data. The XOP format uses multipart MIME to enable raw binary data to be included into an XML 1.0 document without resorting to base64 encoding.

A companion specification, SOAP Message Transmission Optimization Method (MTOM) then specifies how to bind this format to SOAP. The XOP and MTOM standards should enhance SOAP 1.2 performance. XOP and MTOM together provide the preferred approach for mixing binary data with text-based XML. Coupled together, MTOM and XOP enables us to select what parts of the message need to be sent over the wire as binary while still maintaining the Infoset. These standards enable the attachment of binary data outside of the SOAP envelope as a message part. However, unlike SwA, the binary data is treated very much as it was within the SOAP envelope, meaning one Infoset.

XOP defines a serialization mechanism for XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but applicable to any XML Infoset and any packaging mechanism. On the other hand, XML is not a good general-purpose packaging mechanism.

An XOP package is created by placing a serialization of the XML Infoset inside of an extensible packaging format (such a MIME). Note that XOP does reuse MIME for the actual packaging on the wire. Then, selected portions of its content that are base64-encoded binary data are extracted and re-encoded, meaning the data is decoded from base64 and placed into the package. The locations of those selected portions are marked in the XML with a special element that links to the packaged data by using URIs.

The SOAP processing engines performs a temporary Base 64 encoding of the binary data just before the message hits the wire. This temporary encoding enables the SOAP processor to work on the Base 64 data; for example, enabling a WS-Signature of the data to be taken and placed into the header. There is no need for expensive decoding at the other end, and the process works in reverse.

Implementations of MTOM and XOP are available in Java (JAX-WS).

This example shows an XML Infoset prior to XOP processing (SOAP):

```
<soap:Envelope
    xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
    xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo
  xmlmime:contentType='image/png'>/aWKKapGGyQ=</m:photo>
      <m:sig
  xmlmime:contentType='application/pkcs7-signature'>Faa7vROi2VQ=</m:sig>
    </m:data>
  </soap:Body>
</soap:Envelope>
```

This example shows an XML Infoset that is serialized as a XOP package (SOAP)

```
MIME-Version: 1.0
Content-Type: Multipart/Related;boundary=MIME_boundary;
    type="application/xop+xml";
```

```
    start="<mymessage.xml@example.org>";
    startinfo="application/soap+xml; action=\"ProcessData\""
Content-Description: A SOAP message with my pic and sig in it

--MIME_boundary
Content-Type: application/xop+xml;
    charset=UTF-8;
    type="application/soap+xml; action=\"ProcessData\""
Content-Transfer-Encoding: 8bit
Content-ID: <mymessage.xml@example.org>

<soap:Envelope
    xmlns:soap='http://www.w3.org/2003/05/soap-envelope'
    xmlns:xmlmime='http://www.w3.org/2004/11/xmlmime'>
  <soap:Body>
    <m:data xmlns:m='http://example.org/stuff'>
      <m:photo
  xmlmime:contentType='image/png'><xop:Include
    xmlns:xop='http://www.w3.org/2004/08/xop/include'
    href='cid:http://example.org/me.png'/></m:photo>
      <m:sig
  xmlmime:contentType='application/pkcs7-signature'><xop:Include
    xmlns:xop='http://www.w3.org/2004/08/xop/include'
    href='cid:http://example.org/my.hsh'/></m:sig>
    </m:data>
  </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/png
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/me.png>

// binary octets for png

--MIME_boundary
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/my.hsh>

// binary octets for signature

--MIME_boundary--
```

XML information set:

XML Information Set (Infoset) is a World Wide Web Consortium (W3C) specification, dated February 4, 2004. An XML information set is an abstract model of the information that is stored in an XML document. The information set establishes a separation between data and information in a way that suits most common uses of XML. Several of the concrete XML data models are defined by referring to XML information set items and their properties.

Whereas an *XML information set* is an abstract model of the information that is stored in an XML document, an *information item* is an abstract representation of some component of an XML document. SOAP Version 1.2 makes use of this abstraction to define the information in a SOAP message without ever referring to XML Version 1.*x*. The SOAP HTTP binding specifically permits alternative media types that provide for, as a minimum, the transfer of the SOAP XML Infoset.

SOAP Message Transmission Optimization Mechanism (MTOM) describes SOAP 1.2 constructs in terms of information items whereas SOAP 1.1 is defined in terms of XML elements. MTOM enables SOAP bindings to optimize the transmission or wire format (or both) of a SOAP message by selectively encoding portions of the message while still presenting an XML information set to the SOAP application. The SOAP 1.2 attribute is now in the SOAP namespace. The XML information sets require the support of XML

namespaces. The core XML recommendation does not require the support of XML namespaces; however namespaces are required to support the XML schema.

The XML information set does not require or favor a specific interface or class of interfaces. The XML information set specification presents the information set as a tree for the sake of clarity and simplicity, but there is no requirement that the XML information set be made available through a tree structure. Other types of interfaces, including but not limited to event-based and query-based interfaces, are also capable of providing information conforming to the information set. As long as the information in the information set is made available to XML applications in one way or another, the requirements of the XML information set are satisfied.

The XML information set provides a set of definitions to be used in other specifications that refer to the information in a well-formed XML document. For any given XML document, there are a number of corresponding information sets.

- A unique minimal information set consisting of the core properties of the core items and nothing else.
- A unique maximal information set consisting of all the core and all the peripheral items with all the peripheral properties, and one for every combination of present and absent peripheral items and properties in between. The in-between information sets must be fully consistent with the maximal information set.

Information set items

The XML information set is a description of the information that is available in a well-formed XML document, and it describes an abstract data model of an XML document in terms of a set of information set items. An information item is an abstract description of some part of an XML document, and each information item has a set of associated named properties. All other information items are accessible from the properties of the document information item, either directly or indirectly through the properties of other information items.

Guidelines for using information set items include:

- There is no requirement for an XML document to be valid in order to have an information set.
- An XML document has an information set if it satisfies the namespace constraints.
- An XML document has an information set if it is well-formed
- Only one document information item is permited in the information set.
- An information set for an XML document consists of two or more information items.
- The information set for any well-formed XML document will contain at least the minimum information items: one document information item and one element information item.
- Each information item has a set of associated properties, some of which are core and some of which are peripheral.

An information set can contain up to eleven different types of information items:

- Document information item
- Element information items
- Attribute information items
- Processing instruction information items
- Unexpanded entity reference information items
- Character information items
- Comment information items
- The Document Type Declaration (DTD) information item
- Unparsed entity information items
- Notation information items
- Namespace information items

Note that the information set of the XML document might not be a complete list of all information items.

Certain kinds of invalidity affect the values assigned to some properties. Entities, notations, elements and attributes can be undeclared. You can have multiple declarations for notations and elements. Multiple declarations are valid for entities and attributes. An ID can be undefined or multiply defined. Such cases are noted where relevant in the information item definitions in the XML Information Set specification.

Syntax

The XML information set uses a square-bracket syntax, meaning the property names are shown in square brackets. For example, the document information item has the following properties:

Table 4.

Property	Description
[children]	An ordered list of child information items, in document order.
[document element]	The element information item corresponding to the document element.
[notations]	An unordered set of notation information items, one for each notation declared in the DTD. If any notation is multiply declared, this property has no value.
[unparsed entities]	An unordered set of unparsed entity information items, one for each unparsed entity declared in the DTD.
[base URI]	The base URI of the document entity.
[character encoding scheme]	The name of the character encoding scheme in which the document entity is expressed.
[standalone]	An indication of the standalone status of the document, either yes or no. This property is derived from the optional standalone document declaration in the XML declaration at the beginning of the document entity, and has no value if there is no standalone document declaration.
[version]	A string representing the XML version of the document. This property is derived from the XML declaration optionally present at the beginning of the document entity, and has no value if there is no XML declaration.
[all declarations processed]	This property is not strictly speaking part of the information set of the document. Rather it is an indication of whether the processor has read the complete DTD. Its value is a boolean. If it is false, then certain properties (indicated in their descriptions below) might be unknown. If it is true, those properties are never unknown.

All information sets are understood to describe the XML document with all entity references already expanded; that is, represented by the information items corresponding to their replacement text. In the case that an entity reference cannot be expanded, because an XML processor has not read its declaration or its value, explicit provision is made for representing such a reference in the information set.

Differences in SOAP versions:

Both SOAP Version 1.1 and SOAP Version 1.2 are World Wide Web Consortium (W3C) standards. Web services can be deployed that support not only SOAP 1.1 but also support SOAP 1.2. Some changes from SOAP 1.1 that were made to the SOAP 1.2 specification are significant, while other changes are minor.

The SOAP 1.2 specification introduces several changes to SOAP 1.1. This information is not intended to be an in-depth description of all the new or changed features for SOAP 1.1 and SOAP 1.2. Instead, this information highlights some of the more important differences between the current versions of SOAP.

The changes to the SOAP 1.2 specification that are significant include the following updates:

- SOAP 1.1 is based on XML 1.0. SOAP 1.2 is based on XML Information Set (XML Infoset).

  The XML information set (infoset) provides a way to describe the XML document with XSD schema. However, the infoset does not necessarily serialize the document with XML 1.0 serialization on which SOAP 1.1 is based.. This new way to describe the XML document helps reveal other serialization

formats, such as a binary protocol format. You can use the binary protocol format to compact the message into a compact format, where some of the verbose tagging information might not be required.

In SOAP 1.2 , you can use the specification of a binding to an underlying protocol to determine which XML serialization is used in the underlying protocol data units. The HTTP binding that is specified in SOAP 1.2 - Part 2 uses XML 1.0 as the serialization of the SOAP message infoset.

- SOAP 1.2 provides the ability to officially define transport protocols, other than using HTTP, as long as the vendor conforms to the binding framework that is defined in SOAP 1.2. While HTTP is ubiquitous, it is not as reliable as other transports including TCP/IP and MQ.
- SOAP 1.2 provides a more specific definition of the SOAP processing model that removes many of the ambiguities that might lead to interoperability errors in the absence of the Web Services-Interoperability (WS-I) profiles. The goal is to significantly reduce the chances of interoperability issues between different vendors that use SOAP 1.2 implementations.
- SOAP with Attachments API for Java (SAAJ) can also stand alone as a simple mechanism to issue SOAP requests. A major change to the SAAJ specification is the ability to represent SOAP 1.1 messages and the additional SOAP 1.2 formatted messages. For example, SAAJ Version 1.3 introduces a new set of constants and methods that are more conducive to SOAP 1.2 (such as getRole(), getRelay()) on SOAP header elements. There are also additional methods on the factories for SAAJ to create appropriate SOAP 1.1 or SOAP 1.2 messages.
- The XML namespaces for the envelope and encoding schemas have changed for SOAP 1.2. These changes distinguish SOAP processors from SOAP 1.1 and SOAP 1.2 messages and supports changes in the SOAP schema, without affecting existing implementations.
- Java Architecture for XML Web Services (JAX-WS) introduces the ability to support both SOAP 1.1 and SOAP 1.2. Because JAX-RPC introduced a requirement to manipulate a SOAP message as it traversed through the run time, there became a need to represent this message in its appropriate SOAP context. In JAX-WS, a number of additional enhancements result from the support for SAAJ 1.3.
- The Web Services Description Language (WSDL) Version 1.1 specification does not discuss SOAP 1.2. SOAP 1.2 is discussed in the draft versions of WSDL 2.0. WSDL Version 1.1 only defines how to render a SOAP 1.1 payload in a WSDL 1.1 document. To resolve how to represent SOAP 1.2-based services, there is another W3C document that defines how to define a SOAP 1.2 payload within a WSDL 1.1 document. Read about WSDL 1.1 binding extensions for SOAP 1.2.
- SOAP 1.1 is a single document. The SOAP 1.2 specification is organized in the following parts:
  - Part 0 is a non-normative introduction to SOAP.
  - Part 1 describes the structure of SOAP messages, the SOAP processing model and a framework for binding SOAP to underlying protocols. Conformant SOAP implementations must implement everything in Part 1.
  - Part 2 describes optional add-ins to the core of SOAP including a data model and encoding, an RPC convention and a binding to HTTP. Conformant SOAP implementations might implement any of the add-ins in Part 2. However, if add-ins are implemented, they must conform to the relevant parts of the specification.

  A fourth document is the Specification Assertions and Test Collection

SOAP 1.2 has a number of changes in syntax and provides additional, clarified semantics from those that are described in SOAP 1.1. The SOAP 1.2 Primer document lists and describes these syntax changes.

JAX-WS

Java API for XML-Based Web Services (JAX-WS) is the next generation Web services programming model complimenting the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, development of Web services and clients is simplified with more platform independence for Java applications by the use of dynamic proxies and Java annotations.

JAX-WS is a programming model that simplifies application development through support of a standard, annotation-based model to develop Web Service applications and clients. The JAX-WS technology strategically aligns itself with the current industry trend towards a more document-centric messaging model

and replaces the remote procedure call programming model as defined by JAX-RPC. While the JAX-RPC programming model and applications are still supported by this product, JAX-RPC has limitations and does not support various complex document-centric services. JAX-WS is the strategic programming model for developing Web services and is a required part of the Java Platform, Enterprise Edition 5 (Java EE 5). JAX-WS is also known as JSR 224.

Note: WebSphere Application Server Version 7.0 supports the JAX-WS 2.1 specification. JAX-WS 2.1 extends the functionality of JAX-WS 2.0 to provide support for the WS-Addressing in a standardized API. Using this function, you can create, transmit and use endpoint references to target a Web service endpoint. You can use this API to specify the action uniform resource identifiers (URIs) that are associated with the Web Services Description Language (WSDL) operations of your Web service. JAX-WS 2.1 introduces the concept of *features* as a way to programmatically control specific functions and behaviors. There are three standard features: the AddressingFeature for WS-Addressing, the MTOMFeature when optimizing the transmission of binary attachments, and the RespectBindingFeature for wsdl:binding extensions. JAX-WS 2.1 requires Java Architecture for XML Binding (JAXB) Version 2.1 for data binding.

The implementation of the JAX-WS programming standard provides the following enhancements for developing Web services and clients:

- **Better platform independence for Java applications.**

  Using JAX-WS APIs, development of Web services and clients is simplified with better platform independence for Java applications. JAX-WS takes advantage of the dynamic proxy mechanism to provide a formal delegation model with a pluggable provider. This is an enhancement over JAX-RPC, which relies on the generation of vendor-specific stubs for invocation.

- **Annotations**

  JAX-WS introduces support for annotating Java classes with metadata to indicate that the Java class is a Web service. JAX-WS supports the use of annotations based on the Metadata Facility for the Java Programming Language (JSR 175) specification, the Web Services Metadata for the Java Platform (JSR 181) specification and annotations defined by the JAX-WS 2.1 specification. Using annotations within the Java source and within the Java class simplifies development of Web services. Use annotations to define information that is typically specified in deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL files into the source artifacts.

  For example, you can embed a simple @WebService tag in the Java source to expose the bean as a Web service.

  ```
  @WebService

  public class QuoteBean implements StockQuote {

         public float getQuote(String sym) { ... }

  }
  ```

  The @WebService annotation tells the server runtime environment to expose all public methods on that bean as a Web service. Additional levels of granularity can be controlled by adding additional annotations on individual methods or parameters. Using annotations makes it much easier to expose Java artifacts as Web services. In addition, as artifacts are created from using some of the top-down mapping tools starting from a WSDL file, annotations are included within the source and Java classes as a way of capturing the metadata along with the source files.

  Using annotations also improves the development of Web services within a team structure because you do not need to define every Web service in a single or common deployment descriptor as required with JAX-RPC Web services. Taking advantage of annotations with JAX-WS Web services enables parallel development of the service and the required metadata.

  For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS

annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

For example, if your service implementation class for your JAX-WS Web service includes the following:

– the @WebService annotation:

```
@WebService(wsdlLocation="http://myhost.com/location/of/the/wsdl/ExampleService.wsdl")
```

– the webservices.xml file specifies a different file name for the WSDL document as follows:

```
<webservices>
<webservice-description>
<webservice-description-name>ExampleService</webservice-description-name>
<wsdl-file>META-INF/wsdl/ExampleService.wsdl</wsdl-file>
...
</webservice-description>
</webservices>
```

In this case, the value that is specified in the deployment descriptor, META-INF/wsdl/ExampleService.wsdl overrides the annotation value.

- **Invoking Web services asynchronously**

  With JAX-WS, Web services are called both synchronously and asynchronously. JAX-WS adds support for both a polling and callback mechanism when calling Web services asynchronously. Using a polling model, a client can issue a request, get a response object back, which is polled to determine if the server has responded. When the server responds, the actual response is retrieved. Using the callback model, the client provides a callback handler to accept and process the inbound response object. Both the polling and callback models enable the client to focus on continuing to process work without waiting for a response to return, while providing for a more dynamic and efficient model to invoke Web services.

  For example, a Web service interface might have methods for both synchronous and asynchronous requests. Asynchronous requests are identified in bold in the following example:

```
@WebService
public interface CreditRatingService {
      // sync operation
      Score      getCreditScore(Customer customer);
      // async operation with polling
      Response<Score> getCreditScoreAsync(Customer customer);
      // async operation with callback
      Future<?> getCreditScoreAsync(Customer customer,
         AsyncHandler<Score> handler);
}
```

  The asynchronous invocation that uses the callback mechanism requires an additional input by the client programmer. The callback is an object that contains the application code that is run when an asynchronous response is received. Use the following code example to invoke an asynchronous callback handler:

```
CreditRatingService svc = ...;

Future<?> invocation = svc.getCreditScoreAsync(customerFred,
 new AsyncHandler<Score>() {
    public void handleResponse (
        Response<Score> response)
      {
        Score score = response.get();
        // do work here...
      }
   }
);
```

  Use the following code example to invoke an asynchronous polling client:

```
CreditRatingService svc = ...;
Response<Score> response = svc.getCreditScoreAsync(customerFred);

while (!response.isDone()) {
  // Complete an action while we wait.
```

```
}

// No cast needed, because of generics.
Score score = response.get();
```

- **Using resource injection**

  JAX-WS supports resource injection to further simplify development of Web services. JAX-WS uses this key feature of Java EE 5 to shift the burden of creating and initializing common resources in a Java runtime environment from your Web service application to the application container environment, itself. JAX-WS provides support for a subset of annotations that are defined in JSR-250 for resource injection and application life cycle in its runtime environment.

  The application server also supports the usage of the @Resource or @WebServiceRef annotation to declare JAX-WS managed clients and to request injection of JAX-WS services and ports. When either of these annotations are used on a field or method, they result in injection of a JAX-WS service or port instance. The usage of these annotations also results in the type specified by the annotation being bound into the JNDI namespace.

  The @Resource annotation is defined by the JSR-250, Common Annotations specification that is included in Java Platform, Enterprise Edition 5 (Java EE 5). By placing the @Resource annotation on a variable of type javax.xml.ws.WebServiceContext within a service endpoint implementation class, you can request a resource injection and collect the javax.xml.ws.WebServiceContext interface related to that particular endpoint invocation. From the WebServiceContext interface, you can collect the MessageContext for the request associated with the particular method call using the getMessageContext() method.

  The @WebServiceRef annotation is defined by the JAX-WS specification.

  The following example illustrates using the @Resource and @WebServiceRef annotations for resource injection:

```
@WebService
public class MyService {

    @Resource
    private WebServiceContext ctx;

    @Resource
    private SampleService svc;

    @WebServiceRef
    private SamplePort port;

    public String echo (String input) {
        ...
    }

}
```

  Refer to sections 5.2.1 and 5.3 of the JAX-WS specification for more information on resource injection.

- **Data binding with JAXB 2.1**

  JAX-WS leverages the Java Architecture for XML Binding (JAXB) 2.1 API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents. JAXB data binding replaces the data binding described by the JAX-RPC specification.

  The JAXB 2.1 specification provides enhancements such as improved compilation support and introduces support for the @XMLSeeAlso annotation. With the improved compilation support, you now have the flexibility to control the whether a new schema file is generated when using the schemagen schema generator and you can configure the xjc schema compiler so that it does not automatically generate new classes for a particular schema. You can use the @XMLSeeAlso annotation to ensure that JAXB is aware of all the classes included in an inheritance hierarchy for a service endpoint interface.**Dynamic and static clients**

The dynamic client API for JAX-WS is called the dispatch client (javax.xml.ws.Dispatch). The dispatch client is an XML messaging oriented client. The data is sent in either PAYLOAD or MESSAGE mode. When using the PAYLOAD mode, the dispatch client is only responsible for providing the contents of the <soap:Body> and JAX-WS adds the <soap:Envelope> and <soap:Header> elements. When using the MESSAGE mode, the dispatch client is responsible for providing the entire SOAP envelope including the <soap:Envelope>, <soap:Header>, and <soap:Body> elements. JAX-WS does not add anything additional to the message. The dispatch client supports asynchronous invocations using a callback or polling mechanism.

The static client programming model for JAX-WS is the called the proxy client. The proxy client invokes a Web service based on a Service Endpoint interface (SEI), which must be provided.

- **Support for MTOM**

  Using JAX-WS, you can send binary attachments such as images or files along with Web services requests. JAX-WS adds support for optimized transmission of binary data as specified by Message Transmission Optimization Mechanism (MTOM).

- **Multiple data binding technologies**

  JAX-WS exposes the following binding technologies to the end user: XML Source, SOAP Attachments API for Java (SAAJ) 1.3, and Java Architecture for XML Binding (JAXB) 2.1. XML Source enables a user to pass a javax.xml.transform.Source into the runtime environment which represents the data in a Source object to be processed. SAAJ 1.3 now has the ability to pass an entire SOAP document across the interface rather than just the payload itself. This action is done by the client passing the SAAJ SOAPMessage object across the interface. JAX-WS leverages the JAXB 2.1 support as the data binding technology of choice between Java and XML.

- **Support for SOAP 1.2**

  Support for SOAP 1.2 has been added to JAX-WS 2.0. JAX-WS supports both SOAP 1.1 and SOAP 1.2 so that you can send binary attachments such as images or files along with Web services requests. JAX-WS adds support for optimized transmission of binary data as specified by MTOM.

- **Development tools**

  JAX-WS provides the wsgen and wsimport command-line tools for generating portable artifacts for JAX-WS Web services. When creating JAX-WS Web services, you can start with either a WSDL file or an implementation bean class. If you start with an implementation bean class, use the wsgen command-line tool to generate all the Web services server artifacts, including a WSDL file if requested. If you start with a WSDL file, use the wsimport command-line tool to generate all the Web services artifacts for either the server or the client. The wsimport command-line tool processes the WSDL file with schema definitions to generate the portable artifacts, which include the service class, the service endpoint interface class, and the JAXB 2.1 classes for the corresponding XML schema.

- **Support for WS-Addressing (JAX-WS 2.1)**

  JAX-WS 2.1 integrates support for the WS-Addressing standard in to the API. The new API enables you to create, transmit and use endpoint references to target a specific Web service endpoint. You can also explicitly specify the action URIs associated with the Web Services Description Language (WSDL) operations of your Web service.

- **Support for JAX-WS 2.1 features**

  JAX-WS 2.1 introduces the concept of features as a way to programmatically control certain functions or behaviors. There are three standard features: the AddressingFeature , the MTOMFeature, and the RespectBindingFeature. The AddressingFeature is used to enable or disable support for the WS-Addressing Version 1.0 specification. The MTOMFeature is used to enable or disable support for Message Transmission Optimized Mechanism (MTOM) when sending binary attachments. The RespectBindingFeature is used to enable or disable support for wsdl:binding extensions. The application server supports an additional feature, the SubmissionAddressingFeature, which is used to enable or disable support for WS-Addressing Member Submission specification (prior to the WS-Addressing Version 1.0 level of specification).

JAX-WS client programming model:

The Java API for XML-Based Web Services (JAX-WS) Web service client programming model supports both the Dispatch client API and the Dynamic Proxy client API. The Dispatch client API is a dynamic client programming model, whereas the static client programming model for JAX-WS is the Dynamic Proxy client. The Dispatch and Dynamic Proxy clients enable both synchronous and asynchronous invocation of JAX-WS Web services.

- Dispatch client: Use this client when you want to work at the XML message level or when you want to work without any generated artifacts at the JAX-WS level.
- Dynamic Proxy client: Use this client when you want to invoke a Web service based on a service endpoint interface.

Dispatch client

XML-based Web services use XML messages for communications between Web services and Web services clients. The JAX-WS APIs provide high-level methods to simplify and hide the details of converting between Java method invocations and their associated XML messages. However, in some cases, you might desire to work at the XML message level. Support for invoking services at the XML message level is provided by the Dispatch client API. The Dispatch client API, `javax.xml.ws.Dispatch`, is a dynamic JAX-WS client programming interface. To write a Dispatch client, you must have expertise with the Dispatch client APIs, the supported object types, and knowledge of the message representations for the associated Web Services Description Language (WSDL) file. The Dispatch client can send data in either `MESSAGE` or `PAYLOAD` mode. When using the `javax.xml.ws.Service.Mode.MESSAGE` mode, the Dispatch client is responsible for providing the entire SOAP envelope including the `<soap:Envelope>`, `<soap:Header>`, and `<soap:Body>` elements. When using the `javax.xml.ws.Service.Mode.PAYLOAD` mode, the Dispatch client is only responsible for providing the contents of the `<soap:Body>` and JAX-WS includes the payload in a `<soap:Envelope>` element.

The Dispatch client API requires application clients to construct messages or payloads as XML which requires a detailed knowledge of the message or message payload. The Dispatch client supports the following types of objects:

- `javax.xml.transform.Source`: Use `Source` objects to enable clients to use XML APIs directly. You can use `Source` objects with SOAP or HTTP bindings.
- JAXB objects: Use JAXB objects so that clients can use JAXB objects that are generated from an XML schema to create and manipulate XML with JAX-WS applications. JAXB objects can only be used with SOAP or HTTP bindings.
- `javax.xml.soap.SOAPMessage`: Use `SOAPMessage` objects so that clients can work with SOAP messages. You can only use `SOAPMessage` objects with SOAP bindings.
- `javax.activation.DataSource`: Use `DataSource` objects so that clients can work with Multipurpose Internet Mail Extension (MIME) messages. Use `DataSource` only with HTTP bindings.

For example, if the input parameter type is `javax.xml.transform.Source`, the call to the Dispatch client API is similar to the following code example:

```
Dispatch<Source> dispatch = ... create a Dispatch<Source>
Source request = ... create a Source object
Source response = dispatch.invoke(request);
```

The Dispatch parameter value determines the return type of the `invoke()` method.

The Dispatch client is invoked in one of three ways:

- Synchronous invocation for requests and responses using the `invoke` method
- Asynchronous invocation for requests and responses using the `invokeAsync` method with a callback or polling object
- One-way invocation using the `invokeOneWay` methods

Refer to Chapter 4, section 3 of the JAX-WS specification for more information on using a Dispatch client.

Dynamic Proxy client

The static client programming model for JAX-WS is the called the Dynamic Proxy client. The Dynamic Proxy client invokes a Web service based on a Service Endpoint Interface (SEI) which must be provided. The Dynamic Proxy client is similar to the stub client in the Java API for XML-based RPC (JAX-RPC) programming model. Although the JAX-WS Dynamic Proxy client and the JAX-RPC stub client are both based on the Service Endpoint Interface (SEI) that is generated from a WSDL file , there is a major difference. The Dynamic Proxy client is dynamically generated at run time using the Java 5 Dynamic Proxy functionality, while the JAX-RPC-based stub client is a non-portable Java file that is generated by tooling. Unlike the JAX-RPC stub clients, the Dynamic Proxy client does not require you to regenerate a stub prior to running the client on an application server for a different vendor because the generated interface does not require the specific vendor information.

The Dynamic Proxy instances extend the `java.lang.reflect.Proxy` class and leverage the Dynamic Proxy function in the base Java SE Runtime Environment (JRE) 6. The client application can then provide an interface that is used to create the proxy instance while the runtime is responsible for dynamically creating a Java object that represents the SEI.

The Dynamic Proxy client is invoked in one of three ways:

- Synchronous invocation for requests and responses using the `invoke` method
- Asynchronous invocation for requests and responses using the `invokeAsync` method with a callback or polling object
- One-way invocation using the `invokeOneWay` methods

Refer to Chapter 4 of the JAX-WS specification for more information on using Dynamic Proxy clients.

Related concepts

"JAX-WS" on page 353
Java API for XML-Based Web Services (JAX-WS) is the next generation Web services programming model complimenting the foundation provided by the Java API for XML-based RPC (JAX-RPC) programming model. Using JAX-WS, development of Web services and clients is simplified with more platform independence for Java applications by the use of dynamic proxies and Java annotations.

Related tasks

Developing and deploying JAX-WS Web services clients
You can develop Web services clients based on the Web Services for Java Platform, Enterprise Edition (Java EE) specification and the Java API for XML-Based Web Services (JAX-WS) programming model.

Developing a JAX-WS client from a WSDL file
Java API for XML-Based Web Services (JAX-WS) tooling supports generating Java artifacts you need to develop static JAX-WS Web services clients when starting with a Web Services Description Language (WSDL) file.

Invoking JAX-WS Web services asynchronously
Java API for XML-Based Web Services (JAX-WS) provides support for invoking Web services using an asynchronous client invocation. JAX-WS provides support for both a callback and polling model when calling Web services asynchronously. Both the callback model and the polling model are available on the Dispatch client and the Dynamic Proxy client.

Related reference

Web services specifications and APIs

Related information

Java API for XML Web Services (JAX-WS) API documentation

Java API for XML Web Services (JAX-WS) API User's Guide documentation

JAX-WS annotations:

Java API for XML-Based Web Services (JAX-WS) relies on the use of annotations to specify metadata associated with Web services implementations and to simplify the development of Web services. Annotations describe how a server-side service implementation is accessed as a Web service or how a client-side Java class accesses Web services.

The JAX-WS programming standard introduces support for annotating Java classes with metadata that is used to define a service endpoint application as a Web service and how a client can access the Web service. JAX-WS supports the use of annotations based on the Metadata Facility for the Java Programming Language (Java Specification Request (JSR) 175) specification, the Web Services Metadata for the Java Platform (JSR 181) specification and annotations defined by the JAX-WS 2.0 and later (JSR 224) specification which includes JAXB annotations. Using annotations from the JSR 181 standard, you can simply annotate the service implementation class or the service interface and now the application is enabled as a Web service. Using annotations within the Java source simplifies development and deployment of Web services by defining some of the additional information that is typically obtained from deployment descriptor files, WSDL files, or mapping metadata from XML and WSDL into the source artifacts.

Use annotations to configure bindings, handler chains, set names of portType, service and other WSDL parameters. Annotations are used in mapping Java to WSDL and schema, and at runtime to control how the JAX-WS runtime processes and responds to Web service invocations.

For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

Note: In WebSphere Application Server Version 7.0, the default annotation support behavior has changed. In the Version 6.1 Feature Pack for Web services, the default behavior is to scan pre- Java Platform, Enterprise Edition (Java EE) 5 Web application modules to identify JAX-WS services and to scan pre-Java EE 5 Web application modules and EJB modules for service clients during application installation. For Version 7.0, the default behavior is to not scan pre-Java EE 5 modules for annotations during application installation or server startup. You can preserve backward compatibility with the feature pack in either of two ways:

- You can set the `UseWSFEP61ScanPolicy` property in the META-INF/MANIFEST.MF of a WAR file or EJB module to `true`. For example:

  ```
  Manifest-Version: 1.0
  UseWSFEP61ScanPolicy: true
  ```

  When this property is set to `true` in the module's META-INF/MANIFEST.MF file, the module is scanned for JAX-WS annotations regardless of the Java EE version of the module. The default value is `false` and when the default value is in effect, JAX-WS annotations are only supported in modules whose version is Java EE 5 or later.
- You can set the `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` custom Java virtual machine (JVM) property using the administrative console. See the JVM custom properties documentation for the correct navigation path to use. To request annotation scanning in all modules regardless of their Java EE version, set the custom property `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` to `true`. You must change the setting on each server that requires a change in the default behavior.

If the property is set within a module's META-INF/MANIFEST.MF file, this setting takes precedence over the server's custom JVM property. When using either property, you must establish the desired annotation scanning behavior before the application is installed. You cannot dynamically change the scanning behavior once an application is installed. If changes to the behavior are required after your application is installed, you must first uninstall the application, specify the desired scanning behavior using the appropriate property and then install the application again. When federating

nodes that have the `com.ibm.websphere.webservices.UseWSFEP61ScanPolicy` set to `true` in the configuration of the servers contained within the node, this property does not affect the deployment manager. You must set the property to `true` on the deployment manager before the node is federated to preserve the behavior as it was on the node before federation.

Annotations supported by JAX-WS are listed in the table below. The target for annotations is applicable for these Java objects:

- types such as a Java class, enum or interface
- methods
- fields representing local instance variables within a Java class
- parameters within a Java method

Web Services Metadata Annotations (JSR 181)

Annotation class	Annotation	Properties
`javax.jws.WebService`	The **@WebService** annotation marks a Java class as implementing a Web service or marks a service endpoint interface (SEI) as implementing a Web service interface. **Important:** • A Java class that implements a Web service must specify either the @WebService or @WebServiceProvider annotation. Both annotations cannot be present. This annotation is applicable on a client or server SEI or a server endpoint implementation class. • If the annotation references an SEI through the `endpointInterface` attribute, the SEI must also be annotated with the @WebService annotation. • See “Rules for methods on classes annotated with @WebService” on page 379 for additional information.	• Annotation target: Type • Properties: **- name** The name of the `wsdl:portType`. The default value is the unqualified name of the Java class or interface. (String) **- targetNamespace** Specifies the XML namespace of the WSDL and XML elements generated from the Web service. The default value is the namespace mapped from the package name containing the Web service. (String) **- serviceName** Specifies the service name of the Web service: `wsdl:service`. The default value is the simple name of the Java class + *Service*. (String) **- endpointInterface** Specifies the qualified name of the service endpoint interface that defines the services’ abstract Web service contract. If specified, the service endpoint interface is used to determine the abstract WSDL contract. (String) **- portName** The `wsdl:portName`. The default value is WebService.name +*Port* . (String) **- wsdlLocation** Specifies the Web address of the WSDL document defining the Web service. The Web address is either relative or absolute. (String)

Annotation class	Annotation	Properties
`javax.jws.` `WebMethod`	The **@WebMethod** annotation denotes a method that is a Web service operation. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class. **Important:** • The @WebMethod annotation is only supported on classes that are annotated with the @WebService annotation.	• Annotation target: Method • Properties: **- operationName** Specifies the name of the `wsdl:operation` matching this method. The default value is the name of Java method. (String) **- action** Defines the action for this operation. For SOAP bindings, this value determines the value of the SOAPAction header. The default value is the name of Java method. (String) **- exclude** Specifies whether to exclude a method from the Web service. The default value is `false`. (Boolean)
`javax.jws.` `Oneway`	The **@Oneway** annotation denotes a method as a Web service one-way operation that only has an input message and no output message. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • There are no properties on the Oneway annotation.

Annotation class	Annotation	Properties
`javax.jws.` `WebParam`	The **@WebParam** annotation customizes the mapping of an individual parameter to a Web service message part and XML element. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Parameter • Properties: **- name** The name of the parameter. If the operation is remote procedure call (RPC) style and the `partName` attribute is not specified, then this is the name of the `wsdl:part` attribute representing the parameter. If the operation is document style or the parameter maps to a header, then `-name` is the local name of the XML element representing the parameter. This attribute is required if the operation is document style, the parameter style is `BARE`, and the mode is `OUT` or `INOUT`. (String) **- partName** Defines the name of `wsdl:part` attribute representing this parameter. This is only used if the operation is RPC style, or the operation is document style and the parameter style is `BARE`. (String) **- targetNamespace** Specifies the XML namespace of the XML element for the parameter. Applies only for document bindings when the attribute maps to an XML element. The default value is the `targetNamespace` for the Web service. (String) **- mode** The value represents the direction the parameter flows for this method. Valid values are `IN`, `INOUT`, and `OUT`. (String) **- header** Specifies whether the parameter is in a message header rather than a message body. The default value is `false`. (Boolean)

Annotation class	Annotation	Properties
`javax.jws.` `WebResult`	The **@WebResult** annotation customizes the mapping of a return value to a WSDL part or XML element. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: **- name** Specifies the name of the return value as it is listed in the WSDL file and found in messages on the wire. For RPC bindings, this is the name of the `wsdl:part` attribute representing the return value. For document bindings, the `-name` parameter is the local name of the XML element representing the return value. The default value is `return` for RPC and DOCUMENT/WRAPPED bindings. The default value is the method name + `Response` for DOCUMENT/BARE bindings. (String) **- targetNamespace** Specifies the XML namespace for the return value. This parameter is only used if the operation is RPC style or if the operation is DOCUMENT style and the parameter style is BARE. (String) **- header** Specifies whether the result is carried in a header. The default value is `false`. (Boolean) **- partName** Specifies the part name for the result with RPC or DOCUMENT/BARE operations. The default value is `@WebResult.name`. (String)

Annotation class	Annotation	Properties
`javax.jws.HandlerChain`	The **@HandlerChain** annotation associates the Web service with an externally defined handler chain. You can only configure the server side handler by using the @HandlerChain annotation on the Service Endpoint Interface (SEI) or the server endpoint implementation class. Use one of several ways to configure a client side handler. You can configure a client side handler by using the @HandlerChain annotation on the generated service class or SEI. Additionally, you can programmatically register your own implementation of the HandlerResolver interface on the Service, or programmatically set the handler chain on the Binding object.	• Annotation target: Type • Properties: **- file** Specifies the location of the handler chain file. The file location is either an absolute java.net.URL in external form or a relative path from the class file. (String) **- name** Specifies the name of the handler chain in the configuration file. (String)
`javax.jws.SOAPBinding`	The **@SOAPBinding** annotation specifies the mapping of the Web service onto the SOAP message protocol. Apply this annotation to a type or methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class. The method level annotation is limited in what it can specify and is only used if the `style` property is `DOCUMENT`. If the method level annotation is not specified, the **@SOAPBinding** behavior from the type is used.	• Annotation target: Type or Method • Properties: **- style** Defines encoding style for messages sent to and from the Web service. The valid values are `DOCUMENT` and `RPC`. The default value is `DOCUMENT`. (String) **- use** Defines the formatting used for messages sent to and from the Web service. The default value is `LITERAL`. `ENCODED` is not supported. (String) **- parameterStyle** Determines whether the method's parameters represent the entire message body or whether parameters are elements wrapped inside a top-level element named after the operation. Valid values are `WRAPPED` or `BARE`. You can only use the `BARE` value with `DOCUMENT` style bindings. The default value is `WRAPPED`. (String)

JAX-WS Annotations (JSR 224)

Annotation class	Annotation	Properties
`javax.xml.ws.Action`	The **@Action** annotation specifies the WS-Addressing action that is associated with a Web service operation. When you use this annotation with a particular method, and generate the corresponding WSDL document, the WS-Addressing Action extension attribute is added to the input and output elements of the WSDL operation that corresponds to that method. To add this attribute to the WSDL operation, you must also specify the @Addressing annotation on the server endpoint implementation class. If you do not want to use the @Addressing annotation you can supply your own WSDL document with the Action attribute already defined.	• Annotation target: Method • Properties: **- fault** Specifies the array of FaultAction for the wsdl:fault of the operation. (String) **- input** Specifies the action for thewsdl:input of the operation. (String) **- output** Specifies the action for thewsdl:output of the operation. (String)
`javax.xml.ws.BindingType`	The **@BindingType** annotation specifies the binding to use when publishing an endpoint of this type. Apply this annotation to a server endpoint implementation class. **Important:** • You can use the @BindingType annotation on the JavaBeans endpoint implementation class to enable MTOM by specifying either `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING` or `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_MTOM_BINDING` as the value for the annotation.	• Annotation target: Type • Properties: **- value** Indicates the binding identifier Web address. Valid values are `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING`, `javax.xml.ws.soap.SOAPBinding.SOAP12HTTP_BINDING`, and `javax.xml.ws.http.HTTPBinding.HTTP2HTTP_BINDING`. The default value is `javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_BINDING`. (String)

Annotation class	Annotation	Properties
`javax.xml.ws.FaultAction`	The **@FaultAction** annotation specifies the WS-Addressing action that is added to a fault response. This annotation must be contained within an @Action annotation. When you use this annotation with a particular method, the WS-Addressing FaultAction extension attribute is added to the fault element of the WSDL operation that corresponds to that method. To add this attribute to the WSDL operation, you must also specify the @Addressing annotation on the server endpoint implementation class. If you do not want to use the @Addressing annotation you can supply your own WSDL document with the Action attribute already defined.	• Annotation target: Method • Properties: **- value** Specifies the action of the wsdl:fault of the operation. (String) **- output** Specifies the name of the exception class. (String) **- className** Specifies the name of the class representing the request wrapper. (String)
`javax.xml.ws.RequestWrapper`	The **@RequestWrapper** annotation supplies the JAXB generated request wrapper bean, the element name, and the namespace for serialization and deserialization with the request wrapper bean that is used at runtime. When starting with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the `className` attribute is required in this case. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: **- localName** Specifies the local name of the XML schema element representing the request wrapper. The default value is the `operationName` as defined in javax.jws.WebMethod annotation. (String) **- targetNamespace** Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String) **- className** Specifies the name of the class representing the request wrapper. (String)

Annotation class	Annotation	Properties
`javax.xml.ws.ResponseWrapper`	The **@ResponseWrapper** annotation supplies the JAXB generated response wrapper bean, the element name, and the namespace for serialization and deserialization with the response wrapper bean that is used at runtime. When starting with a Java object, this element is used to resolve overloading conflicts in document literal mode. Only the `className` attribute is required in this case. Apply this annotation to methods on a client or server Service Endpoint Interface (SEI) or a server endpoint implementation class.	• Annotation target: Method • Properties: **- localName** Specifies the local name of the XML schema element representing the request wrapper. The default value is the `operationName` + *Response*. `operationName` is defined in javax.jws.WebMethod annotation. (String) **- targetNamespace** Specifies the XML namespace of the request wrapper method. The default value is the target namespace of the SEI. (String) **- className** Specifies the name of the class representing the response wrapper. (String)
`javax.xml.ws.RespectBinding`	The **@RespectBinding** annotation specifies whether the JAX-WS implementation must use the contents of the wsdl:binding for an endpoint. When this annotation is specified, a check is performed to ensure all required WSDL extensibility elements with the enabled attribute set to true are supported. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Method • Properties: **- enabled** Specifies whether the wsdl:binding must be used or not. The default value is `true`. (Boolean)
`javax.xml.ws.ServiceMode`	The **@ServiceMode** annotation specifies whether a service provider needs to have access to an entire protocol message or just the message payload. **Important:** • The @ServiceMode annotation is only supported on classes that are annotated with the @WebServiceProvider annotation.	• Annotation target: Type • Properties: **- value** Indicates whether the provider class accepts the payload of the message, `PAYLOAD` or the entire message `MESSAGE`. The default value is `PAYLOAD`. (String)

Annotation class	Annotation	Properties
`javax.xml.ws.soap.Addressing`	The **@Addressing** annotation specifies that this service wants to enable WS-Addressing support. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Type • Properties: **- enabled** Specifies if WS-Addressing is enabled or not. The default value is `true`. (Boolean) **- required** Specifies that WS-Addressing headers must be present on incoming messages. The default value is `false`. (Boolean)
`javax.xml.ws.soap.MTOM`	The **@MTOM** annotation specifies whether binary content in the body of a SOAP message is sent using MTOM. Apply this annotation to a service endpoint implementation class.	• Annotation target: Class • Properties: **- enabled** Specifies if MTOM is enabled for the JAX-WS endpoint. The default value is `true`. (Boolean) **- threshold** Specifies the minimum size for messages that are sent using MTOM. When the message size is less than this specified integer, the message is inlined in the XML document as base64 or hexBinary data. (integer)

Annotation class	Annotation	Properties
`javax.xml.ws.WebFault`	The **@WebFault** annotation maps WSDL faults to Java exceptions. It is used to capture the name of the fault during the serialization of the JAXB type that is generated from a global element referenced by a WSDL fault message. It can also be used to customize the mapping of service specific exceptions to WSDL faults. This annotation can only be applied to a fault implementation class on the client or server.	• Annotation target: Type • Properties: **- name** Specifies the local name of the XML element that represents the corresponding fault in the WSDL file. The actual value must be specified. (String) **- targetNamespace** Specifies the namespace of the XML element that represents the corresponding fault in the WSDL file. (String) **- faultBean** Specifies the name of the fault bean class. (String)
`javax.xml.ws.WebServiceProvider`	The **@WebServiceProvider** annotation denotes that a class satisfies requirements for a JAX-WS Provider implementation class. **Important:** • A Java class that implements a Web service must specify either the @WebService or @WebServiceProvider annotation. Both annotations cannot be present. • The @WebServiceProvider annotation is only supported on the service implementation class. Any class with the @WebServiceProvider annotation must implement the javax.xml.ws.Provider interface.	• Annotation target: Type • Properties: **- targetNamespace** Specifies the XML namespace of the WSDL and XML elements generated from the Web service. The default value is the namespace mapped from the package name containing the Web service. (String) **- serviceName** Specifies the service name of the Web service: `wsdl:service`. The default value is the simple name of the Java class + *Service*. (String) **- portName** The `wsdl:portName`. The default value is the name of the class + *Port*. (String) **- wsdlLocation** The Web address of the WSDL document defining the Web service. This attribute is required. (String)

Annotation class	Annotation	Properties
`javax.xml.ws.WebServiceRef`	The **@WebServiceRef** annotation defines a reference to a Web service invoked by the client. **Important:** • The @WebServiceRef annotation can be used to inject instances of JAX-WS services and ports. • The @WebServiceRef annotation is only supported in certain class types. Examples are JAX-WS endpoint implementation classes, JAX-WS handler classes, Enterprise JavaBeans classes, and servlet classes. This annotation is supported in the same class types as the @Resource annotation. See the Java Platform, Enterprise Edition (Java EE) 5 specification for a complete list of supported class types.	• Annotation target: Type, Field or Method • Properties: **- name** Specifies the JNDI name of the resource. The field name is the default for field annotations. The JavaBeans property name that corresponds to the method is the default for method annotations. You must specify a value for class annotations as there is no default. (String) **- type** Indicates the Java type of the resource. The field type is the default for field annotations. The type of the JavaBeans property is the default for method annotations. You must specify a value for class annotations as there is no default. (Class) **- mappedName** Specified the name to map this resource to. (String) **- value** Indicates the value of the service class and it is a type that extends `javax.xml.ws.Service`. This attribute is required when the type of the reference is a service endpoint interface. (Class) **- wsdlLocation** The Web address of the WSDL document defining the Web service. This attribute is required. (String)
`javax.xml.ws.WebServiceRefs`	The **@WebServiceRefs** annotation associates multiple @WebServiceRef annotations with a specific class. **Important:** • The @WebServiceRef annotation is only supported in certain class types. Examples are JAX-WS endpoint implementation classes, JAX-WS handler classes, Enterprise JavaBeans classes, and servlet classes. This annotation is supported in the same class types as the @Resource annotation. See the Java Platform, Enterprise Edition (Java EE) 5 specification for a complete list of supported class types.	• Annotation target: Type • Properties: **- value** Specifies an array for multiple Web service reference declarations. This attribute is required.

JAX-WS Common Annotations (JSR 250)

Annotation class	Annotation	Properties
`javax.annotation.Resource`	The **@Resource** annotation marks a WebServiceContext resource needed by the application. **Important:** Applying this annotation to a WebServiceContext type field on the server endpoint implementation class for a JavaBeans endpoint or a Provider endpoint results in the container injecting an instance of the WebServiceContext into the specified field. When this annotation is used in place of the @WebServiceRef annotation, the rules described for the @WebServiceRef annotation apply.	• Annotation target: Field or Method • Properties: **- type** Indicates the Java type of the resource. You are required to use the default, `java.lang.Object` or `javax.xml.ws.Web ServiceContext` value. If the type is the default, the resource must be injected into a field or a method. In this case, the type of the field or the type of the JavaBeans property defined by the method must be `javax.xml.ws.WebServiceContext`. (Class) If you are using this annotation to inject a Web service, see the description of the @WebServiceRef type attribute.
`javax.annotation.Resources`	The **@Resources** annotation associates multiple @Resource annotations with a specific class and serves as a container for multiple resource declarations.	• Annotation target: Field or Method • Properties: **- value** Specifies an array for multiple @Resource annotations. This attribute is required.
`javax.annotation.PostConstruct`	The **@PostConstruct** annotation marks a method that needs to run after dependency injection is performed on the class. Apply this annotation to a JAX-WS application handler, a server endpoint implementation class.	• Annotation target: Method
`javax.annotation.PreDestroy`	The **@PreDestroy** annotation marks a method that must be run when the instance is in the process of being removed by the container. Apply this annotation to a JAX-WS application handler or a server endpoint implementation class.	• Annotation target: Method

IBM proprietary annotations

Annotation class	Annotation	Properties
`com.ibm.websphere.wsaddressing.jaxws21.SubmissionAddressing`	The **@SubmissionAddressing** annotation specifies that this service wants to enable WS-Addressing support for the 2004/08 WS-Addressing specification. This annotation is part of the IBM implementation of the JAX-WS 2.1 specification. Apply this annotation to methods on a server endpoint implementation class.	• Annotation target: Type • Properties: **- required** Specifies that WS-Addressing headers must be present on incoming messages. The default value is `false`. (Boolean)

Rules for methods on classes annotated with @WebService

The following rules apply for methods on classes annotated with the @WebService annotation.

- If the @WebService annotation of an implementation class references an SEI, the implementation class must not have any @WebMethod annotations.
- All public methods for an SEI are considered exposed methods regardless of whether the @WebMethod annotation is specified or not. It is incorrect to have an @WebMethod annotation on an SEI that contains the `exclude` attribute.
- For an implementation class that does not reference an SEI, if the @WebMethod annotation is specified with a value of `exclude=true`, that method is not exposed. If the @WebMethod annotation is not specified, all public methods are exposed including the inherited methods with the exception of methods inherited from java.lang.Object.

JAX-WS application packaging:

You can package a Java Application Programming Interface (API) for XML Web Services (JAX-WS) application as a Web service. A JAX-WS Web service is contained within a Web archive (WAR) file or a WAR module within an enterprise archive (EAR) file.

A JAX-WS enabled WAR file contains:

- A WEB-INF/web.xml file
- Annotated classes that implement the Web services contained in the application module
- [Optional] Web Services Description Language (WSDL) documents that describe the Web services contained in the application module

A WEB-INF/web.xml file is similar to this example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
</web-app>
```

The web.xml might contain servlet or servlet-mapping elements. When customizations to the web.xml file are not needed, the WebSphere Application Server runtime defines them dynamically as the module is loaded. For more information on configuring the web.xml file, read about customizing Web URL patterns in the web.xml file for JAX-WS applications.

Annotated classes must contain, at a minimum, a Web service implementation class that includes the `@WebService` annotation. The definition and specification of the Web services-related annotations are provided by the JAX-WS and JSR-181 specifications. The Web service implementation classes can exist within the `WEB-INF/classes` or directory within a Java archive (JAR) file that is contained in the `WEB-INF/lib` directory of the WAR file.

You can optionally include WSDL documents in the JAX-WS application packaging. If the WSDL document for a particular Web service is omitted, then the WebSphere Application Server runtime constructs the WSDL definition dynamically from the annotations contained in the Web service implementation classes. You must include the `@WebService`, `@WebMethod`, `@WebParam`, `@WebResult`, and optionally the `@SOAPBinding` annotations if the WSDL document is omitted.

Note: In WebSphere Application Server Version 7.0, the default annotation support behavior has changed. In the Version 6.1 Feature Pack for Web services, the default behavior is to scan pre-Java EE 5 Web application modules to identify JAX-WS services and to scan pre-Java EE 5 Web application modules and EJB modules for service clients during application installation. For Version 7.0, the

default behavior is to not scan pre-Java EE 5 modules for annotations during application installation or server startup. You can preserve compatibility with feature packs from previous releases by either setting the UseWSFEP61ScanPolicy property in the META-INF/MANIFEST.MF of a Web archive (WAR) file or EJB module or by defining the Java virtual machine custom property, com.ibm.websphere.webservices.UseWSFEP61ScanPolicy, on servers to request scanning during application installation and server startup. To learn more about annotations scanning, see the JAX-WS annotations documentation.

JAXB

Java Architecture for XML Binding (JAXB) is a Java technology that provides an easy and convenient way to map Java classes and XML schema for simplified development of Web services. JAXB leverages the flexibility of platform-neutral XML data in Java applications to bind XML schema to Java applications without requiring extensive knowledge of XML programming.

JAXB is an XML to Java binding technology that supports transformation between schema and Java objects and between XML instance documents and Java object instances. JAXB consists of a runtime application programming interface (API) and accompanying tools that simplify access to XML documents. JAXB also helps to build XML documents that both conform and validate to the XML schema. Java API for XML-Based Web Services (JAX-WS) leverages the JAXB API and tools as the binding technology for mappings between Java objects and XML documents. JAX-WS tooling relies on JAXB tooling for default data binding for two-way mappings between Java objects and XML documents.

Note: WebSphere Application Server Version 7.0 supports the JAXB 2.1 specification. JAX-WS 2.1 requires JAXB 2.1 for data binding. JAXB 2.1 provides enhancements such as improved compilation support and support for the @XMLSeeAlso annotation. With JAXB 2.1, you can configure the xjc schema compiler so that it does not automatically generate new classes for a particular schema. Similarly, you can configure the schemagen schema generator to not automatically generate a new schema. This enhancement is useful when you are using a common schema and you do not want a new schema generated. JAXB 2.1 also introduces the @XMLSeeAlso annotation that enables JAXB to bind additional Java classes that it might not otherwise know about when binding a Java class with this annotation. This annotation enables JAXB to know about all classes that are potentially involved in marshalling or unmarshalling as it is not always possible or practical to list all of the subclasses of a given Java class. JAX-WS 2.1 also supports the use of the @XMLSeeAlso annotation on a service endpoint interface (SEI) or on a service implementation bean to ensure all of the classes referenced by the annotation are passed to JAXB for processing.

JAXB provides the xjc schema compiler tool, the schemagen schema generator tool, and a runtime framework. You can use the xjc schema compiler tool to start with an XML schema definition (XSD) to create a set of JavaBeans that map to the elements and types defined in the XSD schema. You can also start with a set of JavaBeans and use the schemagen schema generator tool to create the XML schema. Once the mapping between XML schema and Java classes exists, XML instance documents can be converted to and from Java objects through the use of the JAXB binding runtime API. Data stored in XML documents can be accessed without the need to understand the data structure. You can then use the resulting Java classes to assemble a Web services application.

JAXB annotated classes and artifacts contain all the information needed by the JAXB runtime API to process XML instance documents. The JAXB runtime API supports marshaling of JAXB objects to XML and unmarshaling the XML document back to JAXB class instances. Optionally, you can use JAXB to provide XML validation to enforce both incoming and outgoing XML documents to conform to the XML constraints defined within the XML schema.

JAXB is the default data binding technology used by the Java API for XML Web Services (JAX-WS) tooling and implementation within this product. You can develop JAXB objects for use within JAX-WS applications.

You can also use JAXB independently of JAX-WS when you want to leverage the XML data binding technology to manipulate XML within your Java applications.

The following diagram illustrates the JAXB architecture.

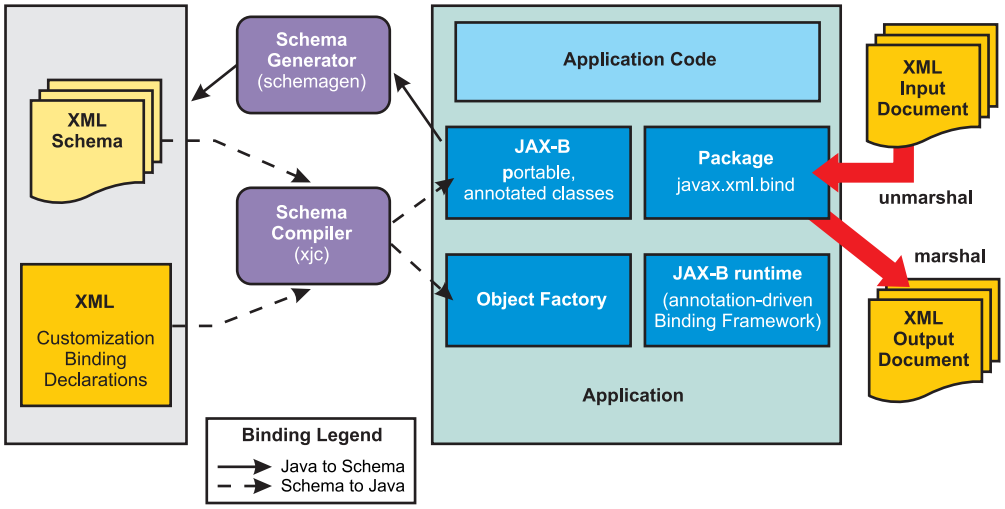


Figure 5. JAXB architecture

JAX-RPC

The *Java API for XML-based RPC (JAX-RPC)* specification enables you to develop SOAP-based interoperable and portable Web services and Web service clients. JAX-RPC 1.1 provides core APIs for developing and deploying Web services on a Java platform and is a part of the Web Services for Java Platform, Enterprise Edition (Java EE) platform. The Java EE platform enables you to develop portable Web services.

WebSphere Application Server implements JAX-RPC 1.1 standards.

The JAX-RPC standard covers the programming model and bindings for using Web Services Description Language (WSDL) for Web services in the Java language. JAX-RPC simplifies development of Web services by shielding you from the underlying complexity of SOAP communication.

On the surface, JAX-RPC looks like another instantiation of remote method invocation (RMI). Essentially, JAX-RPC enables clients to access a Web service as if the Web service was a local object mapped into the client's address space even though the Web service provider is located in another part of the world. The JAX-RPC is done by using the XML-based protocol SOAP, which typically rides on top of HTTP.

JAX-RPC defines the mappings between the WSDL port types and the Java interfaces, as well as between Java language and Extensible Markup Language (XML) schema types.

A JAX-RPC Web service can be created from a JavaBean or a enterprise bean implementation. You can specify the remote procedures by defining remote methods in a Java interface. You only need to code one or more classes that implement the methods. The remaining classes and other artifacts are generated by the Web service vendor's tools. The following is an example of a Web service interface:

```
package com.ibm.mybank.ejb;
import java.rmi.RemoteException;
import com.ibm.mybank.exception.InsufficientFundsException;
/**
 * Remote interface for Enterprise Bean: Transfer
 */
public interface Transfer_SEI extends java.rmi.Remote {
```

```
   public void transferFunds(int fromAcctId, int toAcctId, float amount)
      throws java.rmi.RemoteException;

}
```

The interface definition in JAX-RPC must follow specific rules:

- The interface must extend `java.rmi.Remote` just like RMI.
- Methods must create `java.rmi.RemoteException`.
- Method parameters cannot be remote references.
- Method parameter must be one of the parameters supported by the JAX-RPC specification. The following list are examples of method parameters that are supported. For a complete list of method parameters see the JAX-RPC specification.
  - Primitive types: `boolean`, `byte`, `double`, `float`, `short`, `int` and `long`
  - Object wrappers of primitive types: `java.lang.Boolean`, `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`
  - `java.lang.String`
  - `java.lang.BigDecimal`
  - `java.lang.BigInteger`
  - `java.lang.Calendar`
  - `java.lang.Date`
- Methods can take value objects which consist of a composite of the types previously listed, in addition to aggregate value objects.

A client creates a stub and invokes methods on it. The stub acts like a proxy for the Web service. From the client code perspective, it seems like a local method invocation. However, each method invocation gets marshaled to the remote server. Marshaling includes encoding the method invocation in XML as prescribed by the SOAP protocol.

The following are key classes and interfaces needed to write Web services and Web service clients:

- **Service interface**: A factory for stubs or dynamic invocation and proxy objects used to invoke methods
- **ServiceFactory class**: A factory for Services.
- **`loadService`**

  The `loadService` method is provided in WebSphere Application Server Version 6.0 to generate the service locator which is required by a JAX-RPC implementation. If you recall, in previous versions there was no specific way to acquire a generated service locator. For managed clients you used a JNDI method to get the service locator and for non-managed clients, you were required to instantiate IBM's specific service locator `ServiceLocator service=new ServiceLocator(...);` which does not offer portability. The `loadService` parameters include:
  - **`wsdlDocumentLocation:`** A URL for the WSDL document location for the service or null.
  - **`serviceName`**: A qualified name for the service
  - **`properties`**: A set of implementation-specific properties to help locate the generated service implementation class.
- **`isUserInRole`**

  The `isUserInRole` method returns a boolean indicating whether the authenticated user for the current method invocation on the endpoint instance is included in the specified logical role.
  - **`role`**: The role parameter is a String specifying the name of the role.
- **Service**
- **Call interface**: Used for dynamic invocation
- **Stub interface**: Base interface for stubs

If you are using a stub to access the Web service provider, most of the JAX-RPC API details are hidden from you. The client creates a ServiceFactory (`java.xml.rpc.ServiceFactory`). The client instantiates a Service (`java.xml.rpc.Service`) from the ServiceFactory. The service is a factory object that creates the port. The port is the remote service endpoint interface to the Web service. In the case of DII, the Service object is used to create Call objects, which you can configure to call methods on the Web service's port.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

RMI-IIOP using JAX-RPC:

Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) can be used with JAX-RPC to support non-SOAP bindings.

Java API for XML-based Remote Procedure Call (JAX-RPC) is the Java standard API for invoking Web services through remote procedure calls. A transport is used by a programming language to communicate over the Internet. You can use protocols with the transport such as SOAP and Remote Method Invocation (RMI). You can use Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) with JAX-RPC to support non-SOAP bindings.

Using RMI-IIOP with JAX-RPC, enables WebSphere Java clients to invoke enterprise beans using a WSDL file and the JAX-RPC programming model instead of using the standard Web Services for Java Platform, Enterprise Edition (Java EE) programming model. When a enterprise JavaBeans implementation is used to invoke a Web service, multiprotocol JAX-RPC permits the Web service invocation path to be optimized for WebSphere Java clients. To learn more this optimization, read about using enterprise bean bindings to invoke an EJB from a Web services client.

Benefits of using the RMI/IIOP protocol instead of a SOAP- based protocol are:

- XML processing is not required to send and receive messages; Java serialization is used instead.
- The client JAX-RPC call can participate in a user transaction, which is not the case when SOAP is used.

Web Services-Interoperability Basic Profile

The Web Services-Interoperability (WS-I) Basic Profile is a set of non-proprietary Web services specifications that promote interoperability. WebSphere Application Server conforms to the WS-I Basic Profile Version 1.1 and WS-I Basic Security Profile Version 1.0.

The WS-I Basic Profile is governed by a consortium of industry-leading corporations, including IBM, under direction of the WS-I Organization. The profile consists of a set of principles that relate to bringing about open standards for Web services technology. All organizations that are interested in promoting interoperability among Web services are encouraged to become members of the Web Services Interoperability Organization.

Several technology components are used in the composition and implementation of Web services, including messaging, description, discovery, and security. Each of these components are supported by specifications and standards, including SOAP 1.1, Extensible Markup Language (XML) 1.0, HTTP 1.1, Web Services Description Language (WSDL) 1.1, and Universal Description, Discovery and Integration (UDDI). The WS-I Basic Profile specifies how these technology components are used together to achieve interoperability, and mandates specific use of each of the technologies when appropriate. You can read more about the WS-I Basic Profile at the WS-I Organization Web site.

As technology components are updated, these components are also used in the composition and implementation of Web Services. One example is that both SOAP 1.1 and SOAP 1.2 are now supported.

Note: Building on the support for WS-I Basic Profile Version 1.0, WS-I Basic Profile V1.1, Attachment Profile V1.0, and Basic Security Profile (BSP) V1.0, you can implement Web services with WebSphere Application Server Version 7.0 using the following current emerging standard WS-I profiles:

- *WS-I Basic Profile V1.2* builds on WS-I Basic Profile V1.0 and WS-I Basic Profile V1.1 and adds support for WS-Addressing (WS-A) and SOAP Message Transmission Optimization Mechanism (MTOM). The WS-Addressing specification enables the asynchronous message exchange pattern so that you can decouple the service request from the service response. The SOAP header of the sender's request contains the wsa:ReplyTo value that defines the endpoint reference to which the provider's response is sent. Decoupling the request from the response enables long running Web services interactions. Leveraging the asynchronous programming model support in JAX-WS Version 2.1 in combination with WS-Addressing, you can now take advantage of the ability to create Web services invocations where the client can continue to process work without waiting for a response to return. This provides for a more dynamic and efficient model to invoke Web services. Using MTOM, you can send and receive binary data optimally within a SOAP message.
- *WS-I Basic Profile V2.0* builds on top of Basic Profile V1.2 with the addition of support for SOAP 1.2.
- *WS-I Basic Security Profile V1.1* extends the WS-I Basic Security Profile V1.0 standard by profiling the latest WS-Security V1.1 specification.
- *WS-I Reliable Secure Profile 1.0* builds on WS-I Basic Profile V1.2, WS-I Basic Profile V2.0, WS-I Basic Security Profile V1.0, and WS-I Basic Security Profile V1.1 and adds support for WS-Reliable Messaging 1.1, WS-Make Connection 1.0, and WS-Secure Conversation 1.3. WS-Reliable Messaging 1.1 is a session-based protocol that provides message level reliability for Web services interactions. WS-Make Connection 1.0 was developed by the WS-Reliable Messaging workgroup to address scenarios where a Web services endpoint is behind a firewall or the endpoint has no visible endpoint reference. If a Web services endpoint loses connectivity during a reliable session, WS-Make Connection provides an efficient method to re-establish the reliable session. Additionally, WS-Secure Conversation V1.3 is a session-based security protocol that uses an efficient symmetric key based encryption algorithm for message level security. WS-I Reliable Secure Profile V1.0 provides secure reliable session-oriented Web services interactions.

Each of the technology components has requirements that you can read about in more detail at the WS-I Organization Web site. For example, support for Universal Transformation Format (UTF)-16 encoding is required by WS-I Basic Profile. UTF-16 is a kind of Unicode encoding scheme that uses 16-bit values to store Universal Character Set (UCS) characters. UTF-8 is the most common encoding that is used on the Internet; UTF-16 encoding is typically used for Java and Windows product applications; and UTF-32 is used by various Linux and UNIX systems. Unlike UTF-8, UTF-16 has issues with big-endian and little-endian, and often involves Byte Order Mark (BOM) to indicate the endian. BOM is mandatory for UTF-16 encoding and it can be used in UTF-8.

The application server only supports UTF-8 and UTF-16 encoding of SOAP messages.

The following table summarizes some of the properties of each UTF:

Bytes	Encoding form
EF BB BF	UTF-8
FF FE	UTF-16, little-endian
FE FF	UTF-16, big-endian
00 00 FE FF	UTF-32, big-endian
FF FE 00 00	UTF-32, little-endian

BOM is written prior to the XML text, and it indicates to the parser how the XML is encoded. The XML declaration contains the encoding, for example: <?xml version=*xxx* encoding="utf-*xxx*"?>. BOM is used with the encoding to determine how to interpret the XML. Here is an example of a SOAP message and how BOM and UTF encoding are used:

```
POST http://www.whitemesa.net/soap12/add-test-rpc HTTP/1.1
Content-Type: application/soap+xml; charset=utf-16; action=""
SOAPAction:
Host: localhost: 8080
Content-Length: 562

0xFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
 xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
 xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
 xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <q1:echoString xmlns:q1="http://soapinterop.org/">
   <inputString soap:encodingStyle="http://example.org/unknownEncoding"
   xsi:type="xsd:string">
    Hello SOAP 1.2
   </inputString>
  </q1:echoString>
 </soap:Body>
</soap:Envelope>
```

In the example code, 0xFF0xFE represents the byte codes, while the *<?xml>* declaration is the textual representation.

Support for styleEncoding is not supported in SOAP 1.2 so here is the same example of the SOAP message but without the encoding information:

```
OxFF0xFE<?xml version="1.0" encoding="utf-16"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/12/soap-envelope"
   xmlns:soapenc="http://www.w3.org/2002/12/soap-encoding
 xmlns:tns="http://whitemesa.net/wsdl/soap12-test"
 xmlns:types="http://whitemesa.net/wsdl/soap12-test/encodedTypes"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <q1:echoString xmlns:q1="http://soapinterop.org/">
   <inputString xsi:type="xsd:string">
    Hello SOAP 1.2
   </inputString>
  </q1:echoString>
 </soap:Body>
</soap:Envelope>
```

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

WS-I Attachments Profile

The *Web Services-Interoperability (WS-I) Attachments Profile* is a set of non-proprietary Web services specifications that promote interoperability. This profile compliments the WS-I Basic Profile 1.1 to add support for interoperable SOAP messages with attachments-based Web services.

WebSphere Application Server conforms to the WS-I Attachments Profile 1.0.

Attachments are typically used to send binary data, for example, data that is mapped in Java code to java.awt.Image and javax.activation.DataHandler. The raw data can be sent in the SOAP message, however, this approach is inefficient because an XML parser has to scan the data as it parses the message.

The WS-I Attachments Profile provides a solution to the limitations that are presented by Web Services Description Language (WSDL) 1.1. Because WSDL 1.1 attachments are not part of the XML schema type space, they can be message parts only. As message parts, the attachments cannot be arrays or properties of Java beans. The profile defines the wsi:swaRef XML schema type.Use the wsi:swaRef XML schema type to overcome the limitations of WSDL 1.1 attachments.

The wsi:swaRef type is an extension of the xsd:anyURI type, where its value contains the content-ID of the attachment.

For a complete list of the supported standards and specifications, see the Web services specifications and API documentation.

Web services migration scenarios: JAX-RPC to JAX-WS and JAXB

This topic explains scenarios for migrating your Java API for XML-based RPC (JAX-RPC) Web services to Java API for XML-Based Web Services (JAX-WS) and Java Architecture for XML Binding (JAXB) Web services.

Changes in the tooling can be largely hidden from the user by a development environment like the assembly tools available with WebSphere Application Server. Also, depending on the data specified in the XML, some methods that were used for the JAX-RPC service can be used with little or no change in a JAX-WS service. There are also conditions that do require changes to be made in the JAX-WS service.

Because Java EE environments emphasize compatibility, most application servers that offer support for the newer JAX-WS and JAXB specifications continue to support the previous JAX-RPC specification. A consequence of this is that existing Web services are likely to remain JAX-RPC based while new Web services are developed using the JAX-WS and JAXB programming models.

However, as time passes and applications are revised and rewritten, there might be times when the best strategy is to migrate a JAX-RPC based Web service to one that is based on the JAX-WS and JAXB programming models. This might result from a vendor choosing to provide enhancements to qualities of service that are only available in the new programming models. For example, SOAP 1.2 and SOAP Message Transmission Optimization Mechanism (MTOM) support are only available within the JAX-WS 2.x and JAXB 2.x programming models and not JAX-RPC.

The following information includes issues that you might have while migrating from JAX-RPC to JAX-WS and JAXB.

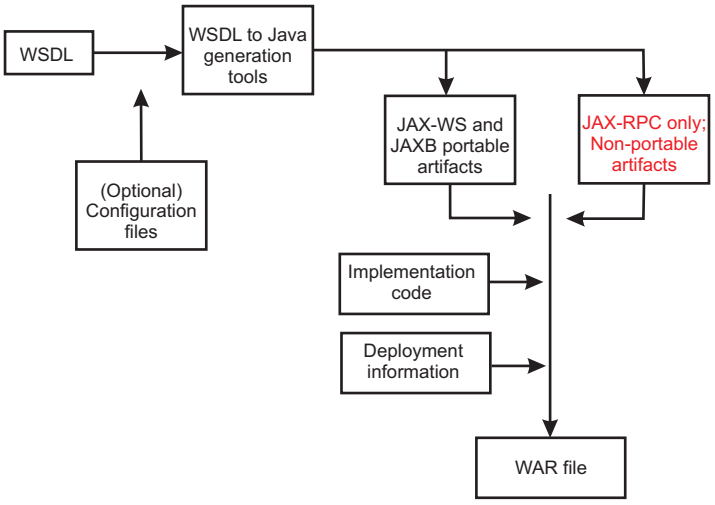
Note: When the terms *migrate*, *migrating*, and *migration* are used in this topic, unless there is some statement to indicate otherwise, the move from a JAX-RPC to a JAX-WS and JAXB environment is being described.

Comparison of JAX-RPC to JAX-WS and JAXB programming models

If you are looking for additional information that describes the changes between the JAX-RPC and JAX-WS and JAXB programming models, IBM developerWorks has published several Web services hints and tips topics.

Development models

The high level combined development models for JAX-RPC and JAX-WS and JAXB are shown the following image:



At a high level, the models are very similar. The only difference is that the JAX-RPC model produces both portable and non-portable artifacts; the JAX-WS and JAXB model does not produce non-portable artifacts.

The image displays the following components:

- **Web Services Description Language (WSDL) file:** A document conforming to the WSDL recommendation as published by the W3C organization.
- **(Optional) Configuration files**
- **WSDL to Java generation tool:** The specifications describe the mappings required, not the name of the tools. Typically, the tool for JAX-WS and JAXB is wsimport, and for JAX-RPC the tools are wscompile and wsdeploy.
- **Portable artifacts:** These are files generated by the WSDL to Java generation tool that have well-defined mappings in the JAX-RPC or JAX-WS and JAXB, whichever is applicable, specifications. These artifacts can be used without modification between different implementations of JAX-RPC or JAX-WS and JAXB.
- **Non-portable artifacts:** These are files generated by the WSDL to Java generation tool that do not have well-defined mappings in the JAX-RPC or JAX-WS and JAXB, whichever is applicable, specifications. These artifacts generally require modification between different implementations of JAX-RPC or JAX-WS/JAXB.
- **Implementation code:** This is the code that you need to meet particular requirements for implementation.
- **Deployment information:** Additional information that is needed to run the application in a particular environment.
- **Web archive (WAR) file:** In the context of the SampleService image, a WAR file is an archive file that contains all items needed to deploy a Web service into a particular environment. This is sometimes called a *cooked WAR file*.

The Development and runtime environments

A specialized development environment simplifies Web services migration by automating many of the tasks. For development of WebSphere-based applications, including Web services, you can use the assembly tools. You can read more about the use of the assembly tools in the assembly tool information center.

The following sample code in this topic was tested in the following runtime environment:

- IBM WebSphere Application Server V6.1, which includes support for the JAX-RPC specification.
- IBM WebSphere Application Server V6.1 Feature Pack for Web Services, which includes support for the JAX-WS and JAXB specifications.

Examples

The following examples show some of the code changes that you might need to migrate your JAX-RPC Web services to JAX-WS and JAXB Web services. In particular, there is an emphasis on the changes in the implementation code that you must write, from both the client and the server side. If no new function is to be introduced, the changes required to move from JAX-RPC to JAX-WS and JAXB might be few.

Sample Web service

The first example is a sample JAX-RPC-based Web Service that was created from a WSDL file (top-down development); the same WSDL file is used to generate the JAX-WS and JAXB-based service. The WSDL file describes the Web service, SampleService, with these operations described by the following image:

As shown in the image, the five operations offered are:

- xsd:int calcShippingCost(xsd:int, xsd:int)
- xsd:string getAccountNumber(xsd:string)
- xsd:dateTime calculateShippingDate(xsd:dateTime)
- xsd:string ckAvailability(xsd:int) creates invalidDateFault
- Person findSalesRep(xsd:string)

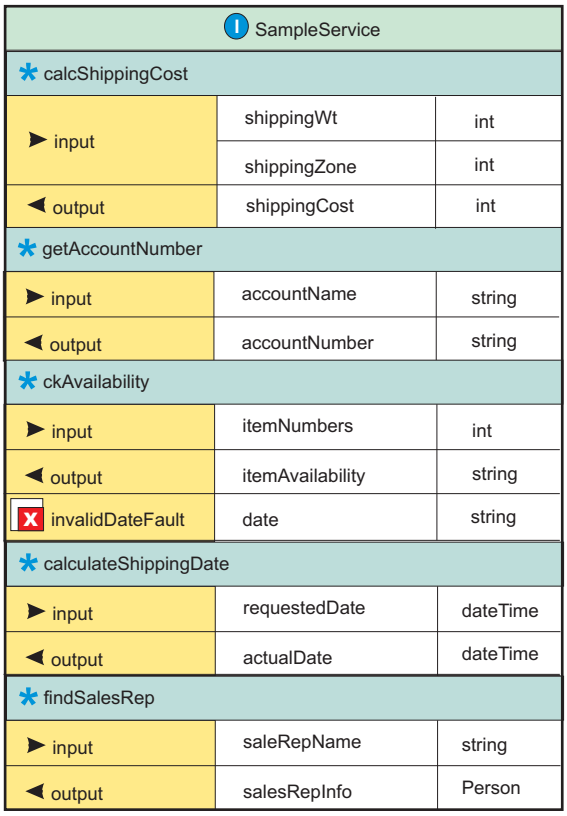
Also assume that Person is defined by the following schema fragment in the WSDL file:

```
<complexType name="Person">
 <sequence>
  <element name="name" type="xsd:string"/>
  <element name="age" type="xsd:int"/>
  <element name="location" type="impl:Address/>
 </sequence>
</complexType>
```

Address is defined by:

```
<complexType name="Address">
 <sequence>
  <element name="street" type="xsd:string"/>
  <element name="city" type="xsd:string"/>
  <element name ="state" type="xsd:string"/>
  <element name="zip" type="xsd:int"/>
 </sequence>
</complexType>
```

The following image also shows that the operation, ckAvailability(xsd:int), which produces an invalidDateFault exception.



Service operations

Review the service code created by the tooling. The following information includes examining what is created for a JAX-RPC runtime and also for a JAX-WS and JAXB runtime.

JAX-RPC

For JAX-RPC, the tooling accepts the WSDL file as input, and, amongst other files, generates the SampleService.java and SampleServiceImpl.java interfaces. The SampleService.java interface defines an interface and the generated code can be review in the following code block. The SampleServiceSoapBindingImpl.java interface provides the skeleton of an implementation, and you typically modify to add your own logic.

JAX-RPC version of SampleService.java:

```
/**
* SampleService.java
*
* This file was auto-generated from WSDL  * by the IBM Web services WSDL2Java emitter.  * cf20633.22 v82906122346
*/
package simple;
public interface SampleService extends java.rmi.Remote {
 public java.lang.Integer calcShippingCost(java.lang.Integer shippingWt,
  java.lang.Integer shippingZone) throws java.rmi.RemoteException;
 public java.lang.String getAccountNumber(java.lang.String accountName)
  throws java.rmi.RemoteException;
 public java.lang.String[] ckAvailability(int[] itemNumbers)
  throws java.rmi.RemoteException, simple.InvalidDateFault;
 public java.util.Calendar calculateShippingDate(
  java.util.Calendar requestedDate)
  throws java.rmi.RemoteException;
 public simple.Person findSalesRep(java.lang.String saleRepName)
  throws java.rmi.RemoteException; }
```

JAX-WS and JAXB

For JAX-WS and JAXB, the tooling accepts the WSDL file as input, and as in the case of JAX-RPC, generates SampleService.java and SampleServiceImpl.java interfaces. As with JAX-RPC, the SampleService.java interface also defines an interface as shown in the following code block. The SampleServiceImpl.java interface provides the skeleton of an implementation, and you typically modify to add your own logic.

Note: The code has been annotated by the tooling.
JAX-WS and JAXB version of the SampleService.java interface:

```
package simple;
 import java.util.List;
 import javax.jws.WebMethod;
 import javax.jws.WebParam;
 import javax.jws.WebResult;
 import javax.jws.WebService;
 import javax.xml.datatype.XMLGregorianCalendar;
 import javax.xml.ws.RequestWrapper;
 import javax.xml.ws.ResponseWrapper;

/**
* This class was generated by the JAX-WS SI.  * JAX-WS RI IBM 2.0_03-06/12/2007 07:44 PM(Raja)-fcs  *
Generated source version: 2.0
*
*/
@WebService(name = "SampleService", targetNamespace = "http://simple") public interface SampleService {

 /**
 *
 * @param shippingWt
 * @param shippingZone
 * @return
 * returns java.lang.Integer
 */
 @WebMethod
 @WebResult(name = "shippingCost", targetNamespace = "")
 @RequestWrapper(localName = "calcShippingCost", targetNamespace = "http://simple",
className = "simple.CalcShippingCost")
 @ResponseWrapper(localName = "calcShippingCostResponse", targetNamespace = "http://simple",
className = "simple.CalcShippingCostResponse")
 public Integer calcShippingCost(
 @WebParam(name = "shippingWt", targetNamespace = "")
 Integer shippingWt,
 @WebParam(name = "shippingZone", targetNamespace = "")
 Integer shippingZone);

 /**
 *
 *
 @param accountName
 * @return
 *     returns java.lang.String
 */
 @WebMethod
 @WebResult(name = "accountNumber", targetNamespace = "")
 @RequestWrapper(localName = "getAccountNumber", targetNamespace = "http://simple",
className = "simple.GetAccountNumber")
 @ResponseWrapper(localName = "getAccountNumberResponse", targetNamespace = "http://simple",
className = "simple.GetAccountNumberResponse")
 public String getAccountNumber(
  @WebParam(name = "accountName", targetNamespace = "")
  String accountName);
 /**
 *
 * @param requestedDate
 * @return
 *     returns javax.xml.datatype.XMLGregorianCalendar
 */
 @WebMethod
 @WebResult(name = "actualDate", targetNamespace = "")
 @RequestWrapper(localName = "calculateShippingDate", targetNamespace = "http://simple",
className = "simple.CalculateShippingDate")
 @ResponseWrapper(localName = "calculateShippingDateResponse", targetNamespace = "http://simple",
className = "simple.CalculateShippingDateResponse")
```

```
 public XMLGregorianCalendar calculateShippingDate(
  @WebParam(name = "requestedDate", targetNamespace = "")
  XMLGregorianCalendar requestedDate);
 /**
 *
 * @param itemNumbers
 * @return
 *     returns java.util.List<java.lang.String>
 * @throws InvalidDateFault_Exception
 */
 @WebMethod
 @WebResult(name = "itemAvailability", targetNamespace = "")
 @RequestWrapper(localName = "ckAvailability", targetNamespace = "http://simple", className = "simple.CkAvailability")
 @ResponseWrapper(localName = "ckAvailabilityResponse", targetNamespace = "http://simple",
className = "simple.CkAvailabilityResponse")
 public List<String> ckAvailability(
  @WebParam(name = "itemNumbers", targetNamespace = "")
  List<Integer> itemNumbers)
  throws InvalidDateFault_Exception
 ;
 /**
 *
 * @param saleRepName
 * @return
 *     returns simple.Person
 */
 @WebMethod
 @WebResult(name = "salesRepInfo", targetNamespace = "")
 @RequestWrapper(localName = "findSalesRep", targetNamespace = "http://simple", className = "simple.FindSalesRep")
 @ResponseWrapper(localName = "findSalesRepResponse", targetNamespace = "http://simple",
className = "simple.FindSalesRepResponse")
 public Person findSalesRep(
 @WebParam(name = "saleRepName", targetNamespace = "")
 String saleRepName);

}
```

Comparing the code examples

At first glance, it might seem that there is little similarity between the interfaces. If you disregard the additional information added by the annotations for JAX-WS and JAXB, the code samples are similar. For example, the calcShippingCost method in the JAX-WS and JAXB version the following lines of code exist:

```
@WebMethod
 @WebResult(name = "shippingCost", targetNamespace = "")
 @RequestWrapper(localName = "calcShippingCost", targetNamespace = "http://simple",
className = "simple.CalcShippingCost")
 @ResponseWrapper(localName = "calcShippingCostResponse", targetNamespace = "http://simple",
className = "simple.CalcShippingCostResponse")
 public Integer calcShippingCost(
  @WebParam(name = "shippingWt", targetNamespace = "")
  Integer shippingWt,
  @WebParam(name = "shippingZone", targetNamespace = "")
  Integer shippingZone);
```

But, if you discard the annotations, the following lines of code are:

```
public Integer calcShippingCost(
        Integer shippingWt,
        Integer shippingZone);
```

These lines are almost identical to what was generated for JAX-RPC. The only difference is that the JAX-RPC code can produce the java.rmi.RemoteException error as follows:

```
public java.lang.Integer calcShippingCost(java.lang.Integer shippingWt,
   java.lang.Integer shippingZone) throws java.rmi.RemoteException;
```

Following this logic, three of the methods have essentially the same signatures:

```
public Integer calcShippingCost(Integer shippingWt, Integer shippingZone)
 public String getAccountNumber(String accountName)
 public Person findSalesRep(String saleRepName)
```

This means that migrating from JAX-RPC to JAX-WS does not directly affect these methods and the original implementation code that is successfully running in the JAX-RPC based environment can probably be used without modification for these methods.

However two of the methods do have different signatures:

For JAX-RPC:

```
public java.util.Calendar calculateShippingDate(
 java.util.Calendar requestedDate)
public java.lang.String[] ckAvailability(int[] itemNumbers)
 throws java.rmi.RemoteException,
 simple.InvalidDateFault
```

JAX-WS and JAXB:

```
public XMLGregorianCalendar calculateShippingDate(
 XMLGregorianCalendar requestedDate)
public List<String> ckAvailability(List<Integer> itemNumbers)
 throws InvalidDateFault_Exception
```

Note:

You might find it easier to compare the skeleton implementation files since the annotations are not present in SampleServiceImpl.java.:

- SampleServiceSoapBindingImpl.java
- SampleServiceImpl.java

The differences in signatures are due to the following reasons:

- Differences mappings from XML names to Java names

  For the calculateShippingDate method, both the input parameter and the return parameter have changed from type java.util.Calendar to type XMLGregorianCalendar. This is because the WSDL specified these parameters to be of type, xsd:dateTime, JAX-RPC maps this data type to java.util.Calendar, while JAX-WS and JAXB maps it to XMLGregorianCalendar.
- Different mappings of arrays from XML to Java

  For the ckAvailability method, the change is due to the data mappings for XML arrays.

  JAX-RPC maps the following WSDL elements:

  ```
  <element maxOccurs="unbounded" name="itemNumbers" type="xsd:int"/>
  <element maxOccurs="unbounded" name="itemAvailability" type="xsd:string"/>
  ```

  The previous elements are mapped to the following Java source:

  ```
  int[] itemNumbers
  java.lang.String[] itemAvailability
  ```

  JAX-WS and JAXB map the following WSDL elements:

  ```
  List<Integer> itemNumbers
  List<String> ckAvailability
  ```
- Different mappings of exceptions

  For the ckAvailability method, the JAX-RPC code generated the following error:

  ```
  simple.InvalidDateFault
  ```

  The JAX-WS code generates the following error:

  ```
  InvalidDateFault_Exception
  ```

  Except for the names, the constructors for these exceptions are different. Therefore, the actual JAX-RPC code that produces the error might be displayed as:

```
throw new InvalidDateFault("this is an InvalidDateFault");
```

For JAX-WS, the code uses a command similar to the following example::

```
throw new InvalidDateFault_Exception( "this is an InvalidDateFault_Exception", new InvalidDateFault());
```

In cases that use exceptions, the code that uses it needs to change.

Migrating the code

Now that the differences between code have been explained, review the original code for the methods that need changing and how to change this code so that it works in a JAX-WS and JAXB environment.

Assume that the original (JAX-RPC) implementation code is similar to following:

```
public java.util.Calendar calculateShippingDate(
     java.util.Calendar requestedDate) throws java.rmi.RemoteException {
// Set the date to the date that was sent to us and add 7 days.
requestedDate.add(java.util.Calendar.DAY_OF_MONTH, 7);

// . . .

return requestedDate;
}
```

You can write the new code to directly use the new types as in the following examples:

```
public XMLGregorianCalendar calculateShippingDate(
XMLGregorianCalendar requestedDate) {
try {
// Create a data type factory.
DatatypeFactory df = DatatypeFactory.newInstance();
// Set the date to the date that was sent to us and add 7 days.
Duration duration = df.newDuration("P7D");
requestedDate.add(duration);

} catch (DatatypeConfigurationException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

// . . .

return requestedDate;
}
```

Migrating the code for the ckAvailability method

When migrating the code for the ckAvailability method, changes were required because of the way that arrays and exceptions are mapped from XML is different between JAX-RPC and JAX-WS. Assume that the original JAX-RPC Web service code is similar to the following example:

```
public java.lang.String[] ckAvailability(int[] itemNumbers)
{

    String[] myString = new String[itemNumbers.length];
    for (int j = 0; j < myString.length; j++) {

            . . .
             if ( ... )

    throw new simple.InvalidDateFault("InvalidDateFault");

            . . .
            if ( . . .)
               myString[j] = "available:  " + jitemNumbers[j] ;
```

```
            else
               myString[j] = "not available:  " + jitemNumbers[j];

    }
    return myString;
}
```

The previous code accepts an int[] as input and returns a String[]. For the JAX-WS and JAXB version, these are List<Integer> and List<String> elements respectively. Processing for these arrays, and discarding the Exception code, the JAX-RPC code is similar to the following:

```
public java.lang.String[] ckAvailability(int[] itemNumbers)
{

    String[] myString = new String[itemNumbers.length];
    for (int j = 0; j < jitemNumbers.length; j++) {


            . . .
            if ( . . .)
               myString[j] = "available:  " + itemNumbers.get(j);
            else
               myString[j] = "not available:  " + itemNumbers.get(j);
    }
    return myString;
}
```

The following JAX-WS and JAXB equivalent code exists using Lists instead of arrays:

```
List <String> ckAvailability(List <Integer> itemNumbers)
{

    ArrayList<String> retList = new ArrayList<String>();
    for (int count = 0; count < itemNumbers.size(); count++) {


            . . .
            if ( . . .)
               retList.add("available:  " + itemNumbers.get(j));
            else
               retList.add("not available:  " + itemNumbers.get(j));
    }
    return retList;
}
```

The differences in the mappings of exceptions from XML to Java forces the JAX-WS code to use. InvalidDateFault_Exception instead of InvalidDateFault.

This means that you must replace `throw new simple.InvalidDateFault("InvalidDateFault");` with some other code. Therefore, the following line is used for the new code:

```
throw new InvalidDateFault_Exception( "test InvalidDateFault_Exception", new InvalidDateFault());
```

The final JAX-WS and JAXB implementation of the method might be similar to the following code:

```
List <String> ckAvailability(List <Integer> itemNumbers)
{

    ArrayList<String> retList = new ArrayList<String>();
    for (int count = 0; count < itemNumbers.size(); count++) {

if ( . . . ) {
   throw new InvalidDateFault_Exception(
   "test InvalidDateFault_Exception",
   new InvalidDateFault());
}
```

```
. . .
if ( . . .)
   retList.add("available:  " + itemNumbers.get(count));
else
   retList.add("not available:  " + itemNumbers.get(count));
   }
   return retList;
}
```

There are multiple ways that you might choose to migrate the code. With practice and an effective development environment, migrating from JAX-RPC to JAX-WS can be straightforward.

Web services migration best practices

Use these Web services migration best practices when migrating Web services applications.

If you have used the Apache SOAP support to develop Web services client applications in WebSphere Application Server Versions 4, 5, or 5.1, you might need to migrate your applications or the security files for your applications. The following table summarizes the Web services specifications supported by the WebSphere products.

WebSphere Application Server Version	Web services specifications supported
4.0	Apache SOAP 2.2
5.0 and 5.0.1	Apache SOAP 2.3
5.0.2 or later	Java 2 Platform, Enterprise Edition (J2EE), also known as (JSR 109)
6.0.x and 6.1	J2EE (JSR 109)
7.0 or later	Web Services for Java Platform, Enterprise Edition (Java EE) 5 also known as JSR 109

Note: The Apache SOAP 2.2 and Apache SOAP 2.3-based implementations that were available in WebSphere Application Server Version 4.0.*x*, 5.0 and 5.0.1 have been deprecated. It is recommended that applications that are using these SOAP implementations migrate to Web Services for Java EE (JSR 109) support that is provided in current WebSphere Application Server versions.

For more information on migrating your Web services, see Migrating Apache SOAP Web services to Web Services to J2EE standards .

It is recommended that new Web services be developed using the Web services for Java EE specification. For more information, read about implementing Web services applications.

Security cannot be directly migrated from SOAP 2.3 to the Java EE standards. After you have migrated your Web services to the Java EE standards, read about securing Web services for Version 6 applications based on WS-Security.

Follow these best practices for the most optimal migration experience:

The application server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model.

Note: Existing JAX-RPC applications wanting to use JAX-WS features must be rewritten using the JAX-WS programming model.

Redeploy existing JAX-RPC Web services after migrating to a new release of the application server

When migrating to a new release of the application server, it is recommended that you redeploy your Web services applications. You should redeploy your Web services application in the new application server environment because of possible changes to the supported levels of Web services specifications and Web services deployment descriptors in each release. To redeploy your Web service, select **Deploy Web Services** in the Install New Application wizard or use the wsdeploy command. To learn more about this process, see the deploying Web services applications onto application servers documentation.

Migrating a Version 5 Java API for XML-based remote procedure call (JAX-RPC) client that uses SOAP over Java Message Service (JMS) to invoke a Web service

A JAX-RPC client that is run on WebSphere Application Server Version 5, can use SOAP over JMS to invoke a Web service that is run on a Version 5 Application Server.

A user ID and password are not required on the target WebSphere MQ queue. After the application server is migrated to Version 6.*x*, and uses the Version 6.*x* default messaging feature, client requests can fail because basic authentication is enabled. The following error message displays when this migration problem occurs:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server with
endpoint <endpoint name> in bus <bus_name> with reason: CWSIT0016E: The user
ID null failed authentication in bus <bus_name>.
```

When the application server is migrated to Version 6.*x*, and the default messaging provider (service integration technologies) is used, and administrative and application security is enabled for the server or the cell, the service integration bus queue destination inherits the security characteristics of the server or the cell by default. If the server or the cell has basic authentication enabled, the client request fails.

The following options are available to solve this problem. The solutions are listed by the level of security that they impose:

- Disable administrative and application security on the main security panel within the administrative console. To disable administrative and application security, click **Security > Global security**. Deselect the **Enable administrative security** and **Enable application security** options.
- Modify the settings for the service integration bus that hosts the queue destination so that the bus security is disabled and the bus does not inherit security characteristics from the server or the cell. This option is equivalent to the level of security that you can configure in Version 5.
- Configure the basic authentication on each client that uses the service.

Migrating Apache SOAP Web services

See Migrating Apache SOAP Web services to Web Services for J2EE standards to learn how to migrate Apache SOAP Web services. This topic explains how to migrate Web services that were developed using Apache SOAP to Web services that are developed based on the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification.

Migrating Web services assembled with early versions of the Application Server Toolkit or Assembly Toolkit

If you are migrating your Web service or Web service components from earlier versions of the Application Server Toolkit or Assembly Toolkit, refer to the following hints and tips to improve your success:

- Secure Web services are not migrated by the J2EE Migration Wizard when Web services are migrated from J2EE 1.3 to J2EE 1.4.
- The migration of secure Web services requires manual steps.

- After the J2EE migration, the secure binding and extension files must be migrated manually to J2EE 1.4 as follows:
  1. Double click on the `webservices.xml` file to open the Web Services editor.
  2. Select the **Binding Configurations** tab to edit the binding file.
  3. Add all the necessary binding configurations under the new sections **Request Consumer Binding Configuration Details** and **Response Generator Binding Configuration Details**.
  4. Select the **Extension** tab to edit the extension file.
  5. Add all the necessary extension configurations under the new sections **Request Consumer Service Configuration Details** and **Response Generator Service Configuration Details**.
  6. Save and exit the editor.

.

WebSphere Application Server roles and goals

A description of several computing roles that members of your organization might perform when working with WebSphere Application Server.

Enterprise architect

The enterprise architect provides overall leadership for all architectural and technological matters with respect to the company's IT environment.

Solution architect

The solution architect designs and coordinates a new solution, application or component with end-to-end responsibility including both hardware and software elements.

The main goal of the solution architect is to design a solution that supports the specification set by the enterprise architect.

System administrator

The system administrator is responsible for managing systems and software, and for installing operating system upgrades and middleware products in many accounts.

The system administrator installs and configures appropriate hardware and software (including middleware) to implement the design provided by the solution architect. Additionally the system administrator monitors and maintains the configured system, modifying and removing previously configured objects as and when required.

Application developer

The application developer creates business applications.

The goal of the application developer is to develop applications that provide the business services described by the solution architect.

Deploying Web services applications onto application servers

After assembling the artifacts required to enable the Web module for Web services into an enterprise archive (EAR) file, you can deploy the EAR file into the application server.

Before you begin

To deploy Java-based Web services, you need an enterprise application, also known as an EAR file that is configured and enabled for Web services.

A Java API for XML-Based Web Services (JAX-WS) application does not require additional bindings and deployment descriptors for deployment whereas a Java API for XML-based RPC (JAX-RPC) Web services application requires you to add additional bindings and deployment descriptors for application deployment. JAX-WS is much more dynamic, and does not require any of the static data generated by the deployment step required for deploying JAX-RPC applications.

For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

Note: In a mixed node cell, you can only target a JAX-WS enabled enterprise beans module to a server using WebSphere Application Server Version 7.0. However, you can target a JAX-WS enabled Web application archives (WAR) module to a server using either WebSphere Application Server Version 7.0 or WebSphere Application Server Version 6.1 Feature Pack for Web Services

You can use the wsdeploy command with JAX-RPC applications to add WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

To install or deploy a JAX-WS application, you only need to install the JAX-WS enabled EAR file. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

Ensure that you have installed the HTTP or Java Message Service (JMS) router module that was generated with the endptEnabler command onto the same target as your Web services enterprise bean JAR files. These HTTP or JMS router modules are included in your Web services application and they need to use the runtime libraries of the application server.

About this task

This task is one of the steps in developing and implementing Web services.

You can use either the administrative console or the **wsadmin** scripting tool to deploy an EAR file. If you are installing an containing Web services by using the **wsadmin** command, specify the **-deployws** option for JAX-RPC applications. If you are installing an application containing Web services by using the administrative console, select **Deploy WebServices** in the Install New Application wizard. For more information about installing applications using the administrative console see Installing a new application.

If your JAX-RPC Web services application was previously deployed with the **wsdeploy** command, it is not necessary to specify Web services deployment during installation.

The following actions deploy the EAR file with the **wsadmin** command:

1. Start *install_root*/bin/wsadmin from a command prompt.
2. Deploy the EAR file.
   - For JAX-WS Web service applications, enter the **$AdminApp install *EARfile* "-usedefaultbindings"** command at the **wsadmin** prompt.
   - For JAX-RPC Web service applications, enter the **$AdminApp install *EARfile* "-usedefaultbindings -deployws"** command at the **wsadmin** prompt.

Results

You have a Web service installed onto your application server.

Note: While installing Web services applications that contain a large number of enterprise beans onto the application server, you might receive out of memory errors. If you receive out of memory errors, increase the heap size of your Java Virtual Machine (JVM). Read about tuning the IBM virtual machine for Java documentation to learn more about tuning the application server environment.

What to do next

You can confirm that the Web services application was deployed by entering the Web service endpoint URL in a browser, then viewing an informative page. The information page contains the following information:

```
{http://webservice.pli.tc.wssvt.ibm.com}RetireWebServices
Hello! This is an Axis2 Web service!
```

The first line of this information is variable, depending on your Web service. The URI in the brackets is the namespace and the string that follows, in this example RetireWebServices, is the name of the port used to access the Web service.

The next step you might want to consider is to apply security to your Web service.

Provide options to perform the Web services deployment settings

Use this panel to specify options for Web services deployment.

This administrative console panel is a step in the application installation and update wizards.

To view this panel, you must select **Deploy Web services** on the **Select installation options** panel.

To view this administrative console page, complete the following steps:

1. Click **Applications > New application >** ***application_path*** .
2. Select the option to **Show all installation options and parameters** .
3. Click **Next** to get to the **Step: Select installation options** panel.
4. Select **Deploy Web service**.
5. Click **Next** to get to the **Step: Provide options to perform the Web services deployment** panel.

You can specify the Web services deployment options on this panel only when installing or updating an application that uses Web services.

The wsdeploy command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the wsdeploy command. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

The options that you specify set parameter values for the **wsdeploy** command. The **wsdeploy** command adds product-specific deployment classes to a Web services-compatible enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs
- Serializers and deserializers
- Implementations of service interfaces

The **wsdeploy** command is run during installation after you click **Finish** on the **Summary** panel of the wizard.

Related tasks

Installing enterprise application files with the console
Installing Java Platform, Enterprise Edition (Java EE) application files consists of placing assembled enterprise application, Web, enterprise bean (EJB), or other installable modules on a server or cluster configured to hold the files. Installed files that start and run properly are considered *deployed*.

Related reference

"wsdeploy command"
This topic explains how to use the **wsdeploy** command-line tool with Web services. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

Enterprise application settings
Use this page to configure an enterprise application.

Deploy Web services option - Classpath

Specifies entries to add to the CLASSPATH when the generated classes are compiled.

To specify the class paths of multiple entries, you need to separate the entries with a semicolon on Windows platforms and on Linux, Unix, and z/OS platforms, you need to use a colon to separate the entries. This is the same separator that is used with the CLASSPATH environment variable.

This option is the same as the **wsdeploy** command parameter `-cp` *`class_path`*.

Data type	String
Default	null

Deploy Web services option - Extension Directories

Specifies a directory that contains zipped or Java archive (JAR) files. All zipped and JAR files in this directory are added to the CLASSPATH used to compile the generated files.

This option is the same as the **wsdeploy** command parameter `-jardir` *`directory`*.

Data type	String
Default	null

wsdeploy command

This topic explains how to use the **wsdeploy** command-line tool with Web services. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

The wsdeploy command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the wsdeploy command. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

The deployment classes that are added by the wsdeploy tool to a Web services-compatible EAR file or a JAR file include:
- Stubs
- Serializers and deserializers
- Implementations of service interfaces

This deployment step must be performed at least once, and can be performed more often. Deployment can be performed separately using the **wsdeploy** command, assembly tools, or when the application is installed. When using the **wsadmin** command for installation, specify the **-deployws** option.

The **wsdeploy** command operates as noted in the following list:
- Each module in the enterprise application or JAR file is examined.
- If the module contains Web services implementations, indicated by the presence of the `webservices.xml` deployment descriptor, the associated Web Services Description Language (WSDL) files are located and the **WSDL2Java** command is run with the role deploy-server option.
- If the module contains Web services clients, indicated by the presence of the client deployment descriptor, the associated WSDL files are located and the **WSDL2Java** command is run with the role deploy-client option.
- The files generated by the **WSDL2Java** command are compiled and repackaged.

See **WSDL2Java** command for more information about the files that are generated for deployment.

When the generated files are compiled, they can reference application-specific classes outside the EAR or JAR file, if the EAR or JAR file is not self-contained. In this case, use either the -jardir or -cp option to specify additional JAR or zip files to be added to CLASSPATH variable when the generated files are compiled.

wsdeploy command syntax

The command syntax is noted in the following example:

`wsdeploy` *`Input_filename Output_filename`* `[options]`

Required options:
- ***Input_filename***

  Specifies the path to the EAR or JAR file to deploy.
- ***Output_filename***

  Specifies the path of the deployed EAR or JAR file. If *output_filename* already exists, it is silently overwritten. The *output_filename* can be the same as the *input_filename*.

Other options:
- **-jardir** *directory*

  Specifies a directory that contains JAR or zip files. All JAR and zip files in this directory are added to the CLASSPATH used to compile the generated files. This option can be specified zero or more times.
- **-cp** *entries*

  Specifies entries to add to the CLASSPATH when the generated classes are compiled. Multiple entries are separated the same as they are in the CLASSPATH environment variable.
- **-codegen**

  Specifies to generate but not compile deployment code. This option implicitly specifies the -keep option.
- **-debug**

  Includes debugging information when compiling, that is, use javac -g to compile.
- **-help**

  Displays a help message and exit.
- **-ignoreerrors**

  Do not stop deployment if validation or compilation errors are encountered.
- **-keep**

  Do not delete working directories containing generated classes. A message is displayed indicating the name of the working directory that is retained.
- **-novalidate**

  Do not validate the Web services deployment descriptors in the input file.
- **-trace**

Displays processing information, including the names of the generated files.

Example The following example illustrates how the options are used with the **wsdeploy** command:

```
wsdeploy x.ear x_deployed.ear -trace -keep
Processing web service module x_client.jar.
Keeping directory: f:\temp\Base53383.tmp for module: x_client.jar.
Parsing XML file:f:\temp\Base53383.tmp\WarDeploy.wsdl
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java.
Done processing module x_client.jar.
```

Messages

- Flag *-f* is not valid.

  Option *f* was not recognized as a valid option.
- Flag *-c* is ambiguous.

  Options can be abbreviated, but the abbreviation must be unique. In this case, the **wsdeploy** command cannot determine which option was intended.
- Flag *-c* is missing parameter *-p*.

  A required parameter for an option is omitted.
- Missing *p* parameter.

  A required option is omitted.

JAX-WS application deployment model

The administration function of the product is enhanced to support installing and deploying Java Application Programming Interface (API) for XML Web Services (JAX-WS) applications like any other WebSphere Application Server applications.

A JAX-WS application is packaged as a Web archive (WAR) file or a WAR module within an Enterprise Archive (EAR) file. The JAX-WS application deployment model is similar to the Java API for XML Remote Protocol Call (JAX-RPC) Web services application model. The main difference between them is that JAX-RPC Web services application requires you to add additional bindings and deployment descriptors for application deployment. A JAX-WS application does not require additional bindings and deployment descriptors for deployment. You can deploy your JAX-WS applications as you would deploy any other WebSphere Application Server application.

JAX-WS Web services is a rewrite of JAX-RPC Web services. The table compares the Web services stack for both JAX-WS and JAX-RPC Web services.

JAX-RPC Web services	JAX-WS Web services
Bindings are proprietary	Bindings are based on the open source Java API for XML Bindings (JAXB)
Parsing is proprietary	Parsing is based on the open source Java Specification Request (JSR) 173
No Java annotations support	Support for Java annotations such as `@WebService`, `@WebMethod`, `@WebParam`, `@WebResult`, and `@SOAPBinding`

JAX-RPC Web services	JAX-WS Web services
During deployment, some deployment descriptor files are created in a JAX-RPC based service and client. The following files are created on the services side, when it is an EJB based Web service and EJB based module: • webservices.xml • <name_of_service>_mapping.xml • ibm-webservices-bnd.xmi • ibm-webservices-ext.xmi When the service is a JavaBeans-based or Web module-based service, the following files and deployment descriptors are required: • webservices.xml • <name_of_service>_mapping.xml • In the web.xml, there is no additional content • ibm-webservices-bnd.xmi • ibm-webservices-ext.xmi The web.xml exists in both EJB and JavaBeans based services. However, there is no additional content added to the file during deployment of a Web service application or module.	For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

Note: In WebSphere Application Server Version 7.0, the default annotation support behavior has changed. In the Version 6.1 Feature Pack for Web services, the default behavior is to scan pre-Java EE 5 Web application modules to identify JAX-WS services and to scan pre-Java EE 5 Web application modules and EJB modules for service clients during application installation. For Version 7.0, the default behavior is to not scan pre-Java EE 5 modules for annotations during application installation or server startup. You can preserve compatibility with feature packs from previous releases by either setting the UseWSFEP61ScanPolicy property in the META-INF/MANIFEST.MF of a Web archive (WAR) file or EJB module or by defining the Java virtual machine custom property, com.ibm.websphere.webservices.UseWSFEP61ScanPolicy, on servers to request scanning during application installation and server startup. To learn more about annotations scanning, see the JAX-WS annotations documentation.

Deploying Web services applications onto application servers

After assembling the artifacts required to enable the Web module for Web services into an enterprise archive (EAR) file, you can deploy the EAR file into the application server.

Before you begin

To deploy Java-based Web services, you need an enterprise application, also known as an EAR file that is configured and enabled for Web services.

A Java API for XML-Based Web Services (JAX-WS) application does not require additional bindings and deployment descriptors for deployment whereas a Java API for XML-based RPC (JAX-RPC) Web services application requires you to add additional bindings and deployment descriptors for application deployment. JAX-WS is much more dynamic, and does not require any of the static data generated by the deployment step required for deploying JAX-RPC applications.

For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file.

You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

Note: In a mixed node cell, you can only target a JAX-WS enabled enterprise beans module to a server using WebSphere Application Server Version 7.0. However, you can target a JAX-WS enabled Web application archives (WAR) module to a server using either WebSphere Application Server Version 7.0 or WebSphere Application Server Version 6.1 Feature Pack for Web Services

You can use the wsdeploy command with JAX-RPC applications to add WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

To install or deploy a JAX-WS application, you only need to install the JAX-WS enabled EAR file. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

Ensure that you have installed the HTTP or Java Message Service (JMS) router module that was generated with the endptEnabler command onto the same target as your Web services enterprise bean JAR files. These HTTP or JMS router modules are included in your Web services application and they need to use the runtime libraries of the application server.

About this task

This task is one of the steps in developing and implementing Web services.

You can use either the administrative console or the **wsadmin** scripting tool to deploy an EAR file. If you are installing an containing Web services by using the **wsadmin** command, specify the **-deployws** option for JAX-RPC applications. If you are installing an application containing Web services by using the administrative console, select **Deploy WebServices** in the Install New Application wizard. For more information about installing applications using the administrative console see Installing a new application.

If your JAX-RPC Web services application was previously deployed with the **wsdeploy** command, it is not necessary to specify Web services deployment during installation.

The following actions deploy the EAR file with the **wsadmin** command:

1. Start *install_root*/bin/wsadmin from a command prompt.
2. Deploy the EAR file.
   - For JAX-WS Web service applications, enter the **$AdminApp install *EARfile* "-usedefaultbindings"** command at the **wsadmin** prompt.
   - For JAX-RPC Web service applications, enter the **$AdminApp install *EARfile* "-usedefaultbindings -deployws"** command at the **wsadmin** prompt.

Results

You have a Web service installed onto your application server.

Note: While installing Web services applications that contain a large number of enterprise beans onto the application server, you might receive out of memory errors. If you receive out of memory errors, increase the heap size of your Java Virtual Machine (JVM). Read about tuning the IBM virtual machine for Java documentation to learn more about tuning the application server environment.

What to do next

You can confirm that the Web services application was deployed by entering the Web service endpoint URL in a browser, then viewing an informative page. The information page contains the following information:

```
{http://webservice.pli.tc.wssvt.ibm.com}RetireWebServices
Hello! This is an Axis2 Web service!
```

The first line of this information is variable, depending on your Web service. The URI in the brackets is the namespace and the string that follows, in this example RetireWebServices, is the name of the port used to access the Web service.

The next step you might want to consider is to apply security to your Web service.

Provide options to perform the Web services deployment settings

Use this panel to specify options for Web services deployment.

This administrative console panel is a step in the application installation and update wizards.

To view this panel, you must select **Deploy Web services** on the **Select installation options** panel.

To view this administrative console page, complete the following steps:

1. Click **Applications > New application >** ***application_path*** .
2. Select the option to **Show all installation options and parameters** .
3. Click **Next** to get to the **Step: Select installation options** panel.
4. Select **Deploy Web service**.
5. Click **Next** to get to the **Step: Provide options to perform the Web services deployment** panel.

You can specify the Web services deployment options on this panel only when installing or updating an application that uses Web services.

The wsdeploy command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the wsdeploy command. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

The options that you specify set parameter values for the **wsdeploy** command. The **wsdeploy** command adds product-specific deployment classes to a Web services-compatible enterprise archive (EAR) file or an application client Java archive (JAR) file. These classes include:

- Stubs
- Serializers and deserializers
- Implementations of service interfaces

The **wsdeploy** command is run during installation after you click **Finish** on the **Summary** panel of the wizard.

Related tasks

Installing enterprise application files with the console
Installing Java Platform, Enterprise Edition (Java EE) application files consists of placing assembled enterprise application, Web, enterprise bean (EJB), or other installable modules on a server or cluster configured to hold the files. Installed files that start and run properly are considered *deployed*.

Related reference

"wsdeploy command" on page 400
This topic explains how to use the **wsdeploy** command-line tool with Web services. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

Enterprise application settings
Use this page to configure an enterprise application.

Deploy Web services option - Classpath

Specifies entries to add to the CLASSPATH when the generated classes are compiled.

To specify the class paths of multiple entries, you need to separate the entries with a semicolon on Windows platforms and on Linux, Unix, and z/OS platforms, you need to use a colon to separate the entries. This is the same separator that is used with the CLASSPATH environment variable.

This option is the same as the **wsdeploy** command parameter `-cp` *`class_path`*.

Data type	String
Default	null

Deploy Web services option - Extension Directories

Specifies a directory that contains zipped or Java archive (JAR) files. All zipped and JAR files in this directory are added to the CLASSPATH used to compile the generated files.

This option is the same as the **wsdeploy** command parameter `-jardir` *`directory`*.

Data type	String
Default	null

wsdeploy command

This topic explains how to use the **wsdeploy** command-line tool with Web services. The **wsdeploy** command adds WebSphere product-specific deployment classes to a Web services-compatible enterprise application enterprise archive (EAR) file or an application client Java archive (JAR) file.

The wsdeploy command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the wsdeploy command. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

The deployment classes that are added by the wsdeploy tool to a Web services-compatible EAR file or a JAR file include:
- Stubs
- Serializers and deserializers
- Implementations of service interfaces

This deployment step must be performed at least once, and can be performed more often. Deployment can be performed separately using the **wsdeploy** command, assembly tools, or when the application is installed. When using the **wsadmin** command for installation, specify the **-deployws** option.

The **wsdeploy** command operates as noted in the following list:
- Each module in the enterprise application or JAR file is examined.
- If the module contains Web services implementations, indicated by the presence of the `webservices.xml` deployment descriptor, the associated Web Services Description Language (WSDL) files are located and the **WSDL2Java** command is run with the role deploy-server option.
- If the module contains Web services clients, indicated by the presence of the client deployment descriptor, the associated WSDL files are located and the **WSDL2Java** command is run with the role deploy-client option.
- The files generated by the **WSDL2Java** command are compiled and repackaged.

See **WSDL2Java** command for more information about the files that are generated for deployment.

When the generated files are compiled, they can reference application-specific classes outside the EAR or JAR file, if the EAR or JAR file is not self-contained. In this case, use either the -jardir or -cp option to specify additional JAR or zip files to be added to CLASSPATH variable when the generated files are compiled.

wsdeploy command syntax

The command syntax is noted in the following example:

`wsdeploy` *`Input_filename Output_filename`* `[options]`

Required options:
- ***Input_filename***

  Specifies the path to the EAR or JAR file to deploy.
- ***Output_filename***

  Specifies the path of the deployed EAR or JAR file. If *output_filename* already exists, it is silently overwritten. The *output_filename* can be the same as the *input_filename*.

Other options:
- **-jardir** *directory*

  Specifies a directory that contains JAR or zip files. All JAR and zip files in this directory are added to the CLASSPATH used to compile the generated files. This option can be specified zero or more times.
- -**cp** *entries*

  Specifies entries to add to the CLASSPATH when the generated classes are compiled. Multiple entries are separated the same as they are in the CLASSPATH environment variable.
- **-codegen**

  Specifies to generate but not compile deployment code. This option implicitly specifies the -keep option.
- **-debug**

  Includes debugging information when compiling, that is, use javac -g to compile.
- **-help**

  Displays a help message and exit.
- **-ignoreerrors**

  Do not stop deployment if validation or compilation errors are encountered.
- **-keep**

  Do not delete working directories containing generated classes. A message is displayed indicating the name of the working directory that is retained.
- **-novalidate**

  Do not validate the Web services deployment descriptors in the input file.
- **-trace**

  Displays processing information, including the names of the generated files.

Example The following example illustrates how the options are used with the **wsdeploy** command:

```
wsdeploy x.ear x_deployed.ear -trace -keep
Processing web service module x_client.jar.
Keeping directory: f:\temp\Base53383.tmp for module: x_client.jar.
Parsing XML file:f:\temp\Base53383.tmp\WarDeploy.wsdl
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java
Generating f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeploy.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\WarDeployLocator.java.
Compiling f:\temp\Base53383.tmp\generatedSource\com\test\HelloWsBindingStub.java.
Done processing module x_client.jar.
```

Messages

- Flag *-f* is not valid.

  Option *f* was not recognized as a valid option.
- Flag *-c* is ambiguous.

  Options can be abbreviated, but the abbreviation must be unique. In this case, the **wsdeploy** command cannot determine which option was intended.
- Flag *-c* is missing parameter *-p*.

  A required parameter for an option is omitted.
- Missing *p* parameter.

  A required option is omitted.

JAX-WS application deployment model

The administration function of the product is enhanced to support installing and deploying Java Application Programming Interface (API) for XML Web Services (JAX-WS) applications like any other WebSphere Application Server applications.

A JAX-WS application is packaged as a Web archive (WAR) file or a WAR module within an Enterprise Archive (EAR) file. The JAX-WS application deployment model is similar to the Java API for XML Remote Protocol Call (JAX-RPC) Web services application model. The main difference between them is that JAX-RPC Web services application requires you to add additional bindings and deployment descriptors for application deployment. A JAX-WS application does not require additional bindings and deployment descriptors for deployment. You can deploy your JAX-WS applications as you would deploy any other WebSphere Application Server application.

JAX-WS Web services is a rewrite of JAX-RPC Web services. The table compares the Web services stack for both JAX-WS and JAX-RPC Web services.

JAX-RPC Web services	JAX-WS Web services
Bindings are proprietary	Bindings are based on the open source Java API for XML Bindings (JAXB)
Parsing is proprietary	Parsing is based on the open source Java Specification Request (JSR) 173
No Java annotations support	Support for Java annotations such as `@WebService`, `@WebMethod`, `@WebParam`, `@WebResult`, and `@SOAPBinding`

JAX-RPC Web services	JAX-WS Web services
During deployment, some deployment descriptor files are created in a JAX-RPC based service and client. The following files are created on the services side, when it is an EJB based Web service and EJB based module: • webservices.xml • <name_of_service>_mapping.xml • ibm-webservices-bnd.xmi • ibm-webservices-ext.xmi When the service is a JavaBeans-based or Web module-based service, the following files and deployment descriptors are required: • webservices.xml • <name_of_service>_mapping.xml • In the web.xml, there is no additional content • ibm-webservices-bnd.xmi • ibm-webservices-ext.xmi The web.xml exists in both EJB and JavaBeans based services. However, there is no additional content added to the file during deployment of a Web service application or module.	For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

Note: In WebSphere Application Server Version 7.0, the default annotation support behavior has changed. In the Version 6.1 Feature Pack for Web services, the default behavior is to scan pre-Java EE 5 Web application modules to identify JAX-WS services and to scan pre-Java EE 5 Web application modules and EJB modules for service clients during application installation. For Version 7.0, the default behavior is to not scan pre-Java EE 5 modules for annotations during application installation or server startup. You can preserve compatibility with feature packs from previous releases by either setting the UseWSFEP61ScanPolicy property in the META-INF/MANIFEST.MF of a Web archive (WAR) file or EJB module or by defining the Java virtual machine custom property, com.ibm.websphere.webservices.UseWSFEP61ScanPolicy, on servers to request scanning during application installation and server startup. To learn more about annotations scanning, see the JAX-WS annotations documentation.

Using the UDDI registry

The Universal Description, Discovery, and Integration (UDDI) registry is a directory for Web services that is implemented using the UDDI specification. It is a component of WebSphere Application Server.

About this task

Throughout this information, the term *UDDI registry* refers to the UDDI registry component that is supplied as part of WebSphere Application Server.

This section describes the UDDI registry and how to use it. An administrator can install, configure, and manage the UDDI registry. A user can use the UDDI registry user interface and the UDDI Application Programming Interfaces (APIs).

1. To learn about the UDDI registry, see the following topics:
   - “Overview of the Version 3 UDDI registry” on page 410
   - “UDDI registry terminology” on page 412
2. To use the UDDI registry, see the following topics:

- "Getting started with the UDDI registry" on page 977
- Using the UDDI registry user interface
- UDDI registry client programming
- "Java API for XML Registries (JAXR) provider for UDDI" on page 448

3. To install, configure, or manage a UDDI registry, see the following topics:
   - "Getting started with the UDDI registry" on page 977
   - "Setting up and deploying a new UDDI registry" on page 460
   - Configuring UDDI registry security
   - "Managing the UDDI registry" on page 988
   - "UDDI registry management interfaces" on page 415
   - "Configuring SOAP API and GUI services for the UDDI registry" on page 986
   - "Applying an upgrade to the UDDI registry" on page 986
   - "Removing and reinstalling the UDDI registry" on page 978
   - Migrating the UDDI registry
4. To resolve any problems with the UDDI registry, see the following topic:
   - UDDI registry troubleshooting

Overview of the Version 3 UDDI registry

The Universal Description, Discovery, and Integration (UDDI) specification defines a way to publish and discover information about Web services. The term *Web service* describes specific business functionality that is exposed by a company, usually through an Internet connection, to allow another company, its subsidiaries, or software program, to use the service.

You can find the UDDI specification on the OASIS UDDI Web page.

The UDDI specification defines a standard for the visibility, reusability, and manageability that are essential for a service-oriented architecture (SOA) registry service.

The UDDI registry is a directory for Web services that is implemented using the UDDI specification. It is a component of WebSphere Application Server.

A critical component of IBM's on-demand service-oriented architecture, the UDDI registry solves the problem of discovery of technical components for an enterprise and its partners by:

- Providing control, flexibility, and confidentiality so that an enterprise can protect its e-business investments
- Increasing efficiency by making it easier to identify technical assets
- Leveraging existing infrastructures

The following example shows how the UDDI registry can be used in a larger enterprise.

A company has a legacy application that provides telephone numbers and human resources (HR) information about employees. This application is turned into a Web service and published to the registry. A developer in the same company wants to write an application for a procurement function that also needs to provide HR information to the supplier. The application needs to give the supplier access to the employee account codes after the employee provides a name or serial number. Before Web services, the developer might be in one of the following situations:

- The developer does not know about the similar application
- The developer knows about the application, but cannot reuse it because of technical barriers
- The developer knows about the application and reuses it, but only after significant time and negotiation

With UDDI, the developer can search for the Web service and reuse the existing technical component in their new application for the supplier in minutes. The developer saves time and gets the application running sooner, thereby increasing efficiency and saving the company time and money. The UDDI registry was the first version 2 standard-compliant UDDI registry for private enterprise work. The UDDI registry in this version has the following characteristics:

- It supports the UDDI Version 3.0 specification, in addition to the Version 1.0 and Version 2.0 standard APIs.
- It leverages the proven, reliable WebSphere Application Server technology.
- It uses a relational database, such as DB2, for its persistent store.

What is new in UDDI Version 3

The main aspects of the UDDI Version 3 specification that are provided with this version of WebSphere Application Server are as follows:

Improved recognition of the importance of private UDDI registries
Private UDDI registries are registries that are installed, owned, managed, and controlled by a separate body such as a department within a company, a company, an industry consortium, or an e-marketplace.

Publisher-assigned keys
The publisher of a UDDI entity can specify its key, rather than the registry automatically assigning a unique key. This means that more human-friendly, URI-based keys can be used, and it makes it easier to manage multiple registries.

UDDI information model improvements
The UDDI data structures are extended, which improves the ability of UDDI to represent businesses and services through metadata.

Security enhancements
Digital signatures provide additional security. Each of the main UDDI entities can be digitally signed, which improves the integrity and trustworthiness of UDDI data.

Ownership transfer APIs
These APIs support the transfer of the ownership of a UDDI entity from one publisher to another.

UDDI policy
You can set policy to define the behavior of a UDDI registry and therefore recognize the different environments in which a UDDI registry is used.

HTTP GET support for UDDI entities
You can use HTTP GET to access XML representations of each of the UDDI data structures. This extends the HTTP GET service beyond the scope for discovery URLs in the UDDI Version 2 specification.

Additional UDDI registry capabilities

The Version 3 UDDI registry in this version of WebSphere Application Server provides the following capabilities that are additional to support for the UDDI Version 3 specification:

Version 2 UDDI inquiry and publish SOAP API compatibility
There is backward compatibility for the Version 1 and Version 2 SOAP inquiry and publish APIs.

UDDI administrative console extension
The WebSphere Application Server administrative console includes a section that administrators can use to manage UDDI-specific aspects of their WebSphere environment. This management includes the ability to set defaults for initialization of the UDDI node, such as its node ID, and to set the UDDI Version 3 policy values.

UDDI registry administrative interface
The Java Management Extensions (JMX) administrative interface enables administrators to manage UDDI-specific aspects of the WebSphere environment programmatically.

Multidatabase support
The UDDI data is persisted to a registry database. The following database products that are supported by WebSphere Application Server are also supported for use as the persistence store for the UDDI registry. For specific details on supported levels, see Detailed system requirements page.

- Apache Derby Version 10.2
- DB2 for iSeries Versions 5.2 and 5.3
- Oracle Versions 9i and 10g

User-defined value set support
You can create your own categorization schemes or value sets. These are in addition to the standard schemes, such as North American Industry Classification System (NAICS), that are provided with the UDDI registry.

UDDI utility tools
You can use UDDI utility tools to import or export entities that use the UDDI Version 2 API.

UDDI user interface
The UDDI user console supports the UDDI Version 3 inquiry and publish APIs.

UDDI Version 3 client
The Java client for UDDI Version 3 handles the construction of raw SOAP requests for the client application. It is a JAX-RPC client and uses Version 3 data types, which are generated from the UDDI Version 3 Web Services Description Language (WSDL) and schema. These data types are serialized or deserialized to the XML, which constitutes the raw UDDI requests.

UDDI Version 2 clients
The following clients for UDDI Version 2 requests are provided:

- UDDI4J. A Java class library for issuing UDDI requests. This client is provided in WebSphere Application Server Version 5 for both UDDI Version 1 requests (uddi4j.jar) and Version 2 requests (uddi4jv2.jar). These class libraries are still supported, as part of the com.ibm.uddi.jar file, but are now both deprecated.
- JAXR. The Java API for XML Registries (JAXR) is a Java client API for accessing UDDI and ebXML registries. WebSphere Application Server provides a JAXR provider for accessing the UDDI registry that conforms to the JAXR 1.0 specification.
- EJB. An Enterprise JavaBeans (EJB) interface for issuing UDDI Version 2 requests. This client is still supported, but is now deprecated.

UDDI registry terminology

Some terms specific to the UDDI registry are explained. Also, the relationship between the versions of the UDDI registry, the Organization for the Advancement of Structured Information (OASIS) specification, and the WebSphere Application Server level are shown.

Throughout the UDDI information in this information center, the directory locations of WebSphere Application Server are referred to as *app_server_root* and *profile_root*.

UDDI Definitions

bindingTemplate
AbindingTemplate is technical information about a service entry point and construction specifications.

businessEntity
A businessEntity is information about the party who publishes information about a family of services.

businessService
A businessService is descriptive information about a particular service.

customized UDDI node
A customized UDDI node is a UDDI node that is initialized with customized settings for the UDDI properties and UDDI policies. In particular, this type kind of node does not have default values for those properties that are read-only after initialization.

Use a customized UDDI node for anything other than simple testing purposes (for which a default UDDI node is enough). To set up a customized UDDI node, see Setting up a customized UDDI node.

When you first start a customized UDDI node, you must set values for certain properties, and then initialize the node (using the administrative console or UDDI administrative interface), before the node is ready to accept UDDI requests. The properties that you must set control characteristics of the UDDI node that cannot be changed after initialization.

An advantage of using a customized UDDI node is that can set these properties to values that are suitable for your environment and usage of UDDI.

After a customized UDDI node is initialized, it is the same as a default UDDI node except that it uses customized UDDI property and policy values.

default UDDI node
A default UDDI node is a UDDI node that is initialized with default settings for the UDDI properties and UDDI policies, including the properties that are read-only after initialization. A default UDDI node is intended for use for testing and to provide a simple way to become familiar with the behavior of the UDDI registry.

You can set up a default UDDI node in two ways. The first is to run the uddiDeploy.jacl script, specifying the 'default' option, in which case the UDDI database will be an Apache Derby database that is created for you automatically.

The second is to create the database yourself, specifying the default option, which for Apache Derby is the DEFAULT parameter when using the UDDIDerbyCreate.jar file, and for DB2 or Oracle the SQL script insert_default_database_indicator.

After a default UDDI node is initialized, it is the same as a customized UDDI node except that it uses default UDDI property and policy values.

policy profile
A policy profile is set of UDDI policies. The default policy profile is the profile created when the default UDDI node is created. In the default policy profile, the nodeID and root key generator are set to read-only and cannot be changed after installation.

publisherAssertion
A publisherAssertion is information about a relationship between two parties, asserted by one or both.

tModel
A tModel (short for technical model) is a data structure representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system.

tModel keys in a service description are a technical ″fingerprint″ that you can use to trace the compatibility origins of a given service. They provide a common point of reference so that you can identify compatible services.

tModels are used to establish the existence of a variety of concepts and to point to their technical definitions. tModels that represent value sets such as category, identifier, and relationship systems

are used to provide additional data to the UDDI core entities to facilitate discovery along a number of dimensions. This additional data is captured in keyedReferences that reside in categoryBags, identifierBags, or publisherAssertions. The tModelKey attributes in these keyedReferences refer to the value set that relates to the concept or namespace being represented. The keyValues contain the actual values from that value set. In some cases, keyNames are significant, for example, to describe relationships and when using the general keywords value set. In all other cases, keyNames provide a version of the keyValue that people can read.

UDDI application
The UDDI application is the UDDI registry enterprise application.

UDDI entitlement
A UDDI entitlement is an entitlement that a UDDI user or publisher has in a UDDI registry, such as the capability to publish keyGenerators, or the tier to which the publisher is assigned (in other words, the number of entities that the publisher is entitled to publish). Each UDDI publisher has a range of settings for the various UDDI entitlements. A UDDI entitlement is sometimes referred to as a 'user entitlement', or as the UDDI publisher's set of 'user entitlements'.

UDDI node
A UDDI node is a set of Web Services that support at least one of the UDDI API sets, which supports interaction with UDDI data through the UDDI APIs. There is no direct mapping between a UDDI node and a WebSphere Application Server node. A UDDI node consists of an instance of the UDDI application running in an application server (or a cluster of UDDI application instances running in a cluster of application servers), together with an instance of the UDDI database containing UDDI data.

UDDI node initialization
UDDI node initialization is the process that sets up values in the UDDI database, and establishes the ″personality″ of the UDDI node. A UDDI node cannot accept UDDI API requests until it is initialized.

UDDI node state
The UDDI node state describes the current state of the UDDI node, as opposed to the state of the UDDI application (which is either stopped or started). A UDDI node can be in one of the following states:

- not initialized
- initialization pending
- initialization in progress
- migration pending
- migration in progress
- value set creation pending
- value set creation in progress
- activated
- deactivated

UDDI NodeId
The UDDI NodeId is a unique identifier of a UDDI node.

UDDI policy
A UDDI policy is a statement of required and expected behavior of a UDDI registry, specified using policy values for the various policies that are defined in the UDDI Version specification.

UDDI property
A UDDI property is a value for a property that controls the personality or behavior of a UDDI node.

UDDI publisher
A UDDI publisher is a WebSphere user who is entitled to publish UDDI entities to a specified UDDI registry. A UDDI publisher is sometimes referred to as a 'UDDI user', or simply as a 'publisher' when used in a UDDI context.

UDDI registry
A UDDI registry comprises one or more UDDI nodes. The UDDI registry in this version of WebSphere Application Server supports single-node UDDI registries only.

UDDI tier
A UDDI tier determines the number of UDDI entities of each type (business, services per business, bindings per service, tModel, publisher assertion) that a UDDI publisher is entitled to publish. Each UDDI publisher is assigned (either by default or explicitly by a UDDI administrator) to a particular tier, and cannot publish more entities than are allowed for that tier. There are some predefined tiers supplied with the UDDI registry, and a UDDI administrator can create additional tiers. A UDDI tier is often referred to simply as a 'tier' when used in a UDDI context.

Version 2 UDDI registry
A Version 2 UDDI registry is a UDDI registry implementation that supports Version 2 of the UDDI specification and also Version 1. A Version 2 UDDI registry is included in WebSphere Application Server Network Deployment Version 5.x.

Version 3 UDDI registry
A Version 3 UDDI registry is a UDDI registry implementation that supports Version 3 of the UDDI specification, and also Versions 1 and 2. A Version 3 UDDI registry is included in WebSphere Application Server. Note that Version 3 UDDI registry does not indicate a UDDI registry implementation that supports only UDDI Version 3 requests.

The following table shows how the various versions of the UDDI registry relate to the relevant OASIS specification and WebSphere Application Server level:

UDDI registry Version	OASIS UDDI specification levels supported	Supported on WebSphere Application Server version
1.1	• UDDI Version 1 • UDDI Version 2	4.0.2
1.1.1	• UDDI Version 1 • UDDI Version 2	4.0.3 and later
2.0.x	• UDDI Version 1 • UDDI Version 2	5.0.x
2.1.x	• UDDI Version 1 • UDDI Version 2	5.1.x
3.0.2	• UDDI Version 1 • UDDI Version 2.0.4 (APIs), Version 2.0.3 (data structures) • UDDI Version 3.0.2	6.0 and later

UDDI registry management interfaces

This topic explains interfaces and tools that you can use to manage UDDI nodes programmatically.

UDDI registry Administrative (JMX) Interface

The UDDI registry Administrative (JMX) Interface provides a Java API that allows you to manage runtime configuration settings to control UDDI registry runtime behavior, such as setting the maximum number of

results that UDDI users can receive for inquiry requests, or creating publish limits for UDDI publishers. Sample client code is provided for you to build on.

User Defined Value Set Support in the UDDI registry

User Defined Value Set Support in the UDDI registry explains the tooling provided to manage your own categorization value sets, including loading value set data into a UDDI registry node.

UDDI Utility Tools

UDDI Utility Tools explains the tooling and Java API for promoting version 2 entities from one UDDI registry to another while retaining entity keys. This is particularly useful for publishing canonical tModels with a predefined key.

UDDI registry Administrative (JMX) Interface

You can use the UDDI registry Administrative Interface to inspect and manage the runtime configuration of a UDDI application. You can manage the information and the activation state about a UDDI node, update properties and policies, set publish tier limits, register UDDI publishers, and control value set support.

The operations of the UDDI registry Administrative Interface can be read and invoked using standard JMX (Java Management Extensions) interfaces. The use of JMX is explained in Using administrative programs (JMX).

Each WebSphere UDDI registry application registers an MBean with an MBean identifier of 'UddiNode'. This MBean may be used by client applications to inspect and manage the runtime configuration of a UDDI application. This includes managing the activation state of and information about a UDDI node, updating properties and policies, setting publish tier limits, registration of UDDI publishers, and controlling value set support.

You can read and invoke the UddiNode attributes and operations using standard JMX interfaces. A client utility class UddiNodeProxy.java provides a ready-made application to connect to a UddiNode MBean and perform all the available operations. Example classes are also provided to drive UddiNodeProxy and demonstrate how to use the various UDDI management data types.

When WebSphere Application Server security is enabled, you can only invoke the operations of the UddiNode MBean if you are a user in an administrative role. The operations which make updates require the Administrator or Operator role, while get operations can be performed by Administrator, Operator, Configurator and Monitor roles.

UddiNodeProxy Usage

The following .jar files are required for compilation:

- *app_server_root*/plugins/com.ibm.uddi.jar
- *app_server_root*/runtimes/com.ibm.ws.admin.client.jar

The UddiNodeProxy class provides a utility method to programmatically interrogate the UddiNode MBean and output all the available attributes, operations and notifications to System.out. For each operation, the return type, operation name and parameter types are output as well as the impact property which indicates how the operation changes the state of the UddiNode MBean (and the UDDI node). As for all MBeans, the value for the impact property can be one of:

ACTION:
state of MBean will be changed

INFO: of the MBean remains unchanged and will return information

ACTION_INFO:
state of the MBean will change and return some information

UNKNOWN:
the impact of invoking the operation is not known

1. Invoke outputMBeanInterface:`uddiNode.outputMBeanInterface();`

   Expected output:

```
java.lang.String getNodeID() [INFO]
(getter for attribute nodeID)
java.lang.String getNodeState() [INFO]
(getter for attribute nodeState)
java.lang.String getNodeDescription() [INFO]
(getter for attribute nodeDescription)
java.lang.String getNodeApplicationName() [INFO]
(getter for attribute nodeApplicationName)
void activateNode() [ACTION]
(activates UDDI node)
void deactivateNode() [ACTION]
(deactivates UDDI node)
void initNode() [ACTION]
(initializes Uddi node)
com.ibm.uddi.v3.management.Property getProperty(java.lang.String propertyId) [INFO]
(returns UDDI Property)
com.ibm.uddi.v3.management.PolicyGroup getPolicyGroup(java.lang.String policyGroupId) [INFO]
(returns UDDI PolicyGroup)
com.ibm.uddi.v3.management.Policy getPolicy(java.lang.String policyId) [INFO]
(returns UDDI Policy)
void updatePolicy(com.ibm.uddi.v3.management.Policy policy) [ACTION]
(updates UDDI Policy)
void updateProperty(com.ibm.uddi.v3.management.ConfigurationProperty property) [ACTION]
(updates UDDI Property)
void updateProperties(java.util.List properties) [ACTION]
(updates collection of UDDI properties)
void updatePolicies(java.util.List policies) [ACTION]
(updates collection of UDDI policies)
java.util.List getProperties() [INFO]
(returns the collection of UDDI properties)
java.util.List getPolicyGroups() [INFO]
(returns collection of policy groups (note that the policies are not populated))
java.util.List getValueSets() [INFO]
(returns collection of value set status objects)
com.ibm.uddi.v3.management.ValueSetStatus getValueSetDetail(java.lang.String tModelKey) [INFO]
(returns status for a value set)
com.ibm.uddi.v3.management.ValueSetProperty getValueSetProperty(java.lang.String tModelKey,java.lang.
String valueSetPropertyId) [INFO]
(returns a property of a value set)
void updateValueSet(com.ibm.uddi.v3.management.ValueSetStatus valueSet) [ACTION]
(updates value set status)
void updateValueSets(java.util.List valueSets) [ACTION]
(updates multiple value sets)
void loadValueSet(java.lang.String filePath,java.lang.String tModelKey) [ACTION]
(loads values for a value set from a UDDI registry V3/V2 taxonomy data file.)
void loadValueSet(com.ibm.uddi.v3.management.ValueSetData valueSetData) [ACTION]
(loads values for a value set with the given tModel key.)
void changeValueSetTModelKey(java.lang.String oldTModelKey,java.lang.String newTModelKey) [ACTION]
(replaces all occurrences of values belonging to original tModelKey to new tModelKey.)
void unloadValueSet(java.lang.String tModelKey) [ACTION]
(unloads values for a value set with the given tModel key.)
java.lang.Boolean isExistingValueSet(java.lang.String tModelKey) [INFO]
(Determine if Value Set data exists for the given tModel key.)
java.util.List getTierInfos() [INFO]
(returns the collection of UDDI tier descriptions.)
java.util.List getLimitInfos() [INFO]
(returns the collection of UDDI limit descriptions.)
java.util.List getEntitlementInfos() [INFO]
```

```
(returns the collection of UDDI entitlements.)
com.ibm.uddi.v3.management.Tier getTierDetail(java.lang.String tierId) [INFO]
(returns UDDI Tier detail, specifying limits to the number of entities that can be published.)
com.ibm.uddi.v3.management.Tier createTier(com.ibm.uddi.v3.management.Tier tier) [ACTION]
(creates a UDDI Tier, specifying limits to the number of entities that can be published. Returns the
new tier ID.)
com.ibm.uddi.v3.management.Tier updateTier(com.ibm.uddi.v3.management.Tier tier) [ACTION]
(updates UDDI Tier details. Returns the updated Tier.)
void deleteTier(java.lang.String tierId) [ACTION]
(deletes the UDDI Tier, if it not in use.)
void setDefaultTier(java.lang.String tierId) [ACTION]
(Specifies the tier that auto registered UDDI publishers are assigned to.)
java.lang.Integer getUserCount(java.lang.String tierId) [INFO]
(returns the number of UDDI publisher within the specified tier.)
com.ibm.uddi.v3.management.TierInfo getUserTier(java.lang.String userId) [INFO]
(returns UDDI Tier information, specifying the tier this user belongs to.)
com.ibm.uddi.v3.management.UddiUser getUddiUser(java.lang.String userId) [INFO]
(returns UDDI user details, including tier and entitlements details.)
java.util.List getUserInfos() [INFO]
(returns the collection of UDDI user names and the tier they belong to.)
void createUddiUser(com.ibm.uddi.v3.management.UddiUser user) [ACTION]
(creates a new UDDI user.)
void createUddiUsers(java.util.List users) [ACTION]
(creates the collection of new UDDI users.)
void updateUddiUser(com.ibm.uddi.v3.management.UddiUser user) [ACTION]
(updates UDDI user details.)
void deleteUddiUser(java.lang.String userId) [ACTION]
(deletes UDDI publisher.)
void assignTier(java.util.List userIds,java.lang.String tierId) [ACTION]
(sets the tier for a List of users.)
notificationInfo: description=default UDDI event,descriptorType=notification,severity=(6),name=
uddi.node.event
notificationInfo: description=null,descriptorType=notification,severity=(6),name=jmx.attribute.
changed
```

See ManageNodeInfoSample class for sample code that demonstrates the attributes and operations described in this section.

Managing UDDI Node States and Attributes

UDDI nodes can be in one of several states, depending on the way the UDDI application was installed (as a default configuration or one where the administrator controls when initialization occurs). The UddiNode MBean provides four read only attributes: nodeID, nodeState, nodeDescription and nodeApplicationName. In addition the following MBean operations change UDDI node state: activateNode, deactivateNode and initNode.

nodeID

The node ID is the unique identifier for a UDDI node. If the UDDI application is installed as a default configuration the node ID is automatically generated. If the UDDI application is set up manually, the node ID is set by the administrator. It must be a valid UDDI key.

```
String nodeID = uddiNode.getNode();

System.out.println("node ID: " + nodeId);
```

nodeState

The nodeState attribute can have one of the following values:

nodeState value	English text associated with state
node.state.uninitialized	Not initialized

node.state.initialized	Initialized
node.state.initPending	Initialization pending
node.state.initInProgress	Initialization in progress
node.state.initMigrationPending	Migration pending
node.state.initMigration	Migration in progress
node.state.initValueSetCreationPending	Value set creation pending
node.state.initValueSetCreation	Value set creation in progress
node.state.activated	Activated
node.state.deactivated	Deactivated
node.state.unknown	Unknown

After installing a UDDI application using the default configuration, the UDDI node will be in activated state, that is, ready to receive and process UDDI API requests. The node ID and root key generator and some other properties are generated and cannot be changed. For a manually installed UDDI application where you want to specify the UDDI node ID and root key generator values, starting the UDDI application will put the UDDI node into initPending state. In this state, you can update all writable values up until the point you invoke the initNode operation. The initNode operation loads base tModels and value set data and writes all the configuration data to the UDDI node's database. During initialization the state is initInProgress. When initialization completes, the state changes momentarily to initialized and settles at activated. At this point the state can only be switched between activated and deactivated using deactivateNode and activateNode MBean operations.

Each node state value is in fact a message key which can be looked up in the messages.properties resource bundle. The attribute value can be retrieved using the getNodeState method of UddiNodeProxy:

1. Invoke getNodeState:

```
String nodeStateKey = uddiNode.getNodeState();
```

2. Look up translated text from ResourceBundle and output:

```
String messages = "com.ibm.uddi.v3.management.messages";

ResourceBundle bundle = ResourceBundle.getBundle(messages,
                                                 Locale.ENGLISH);

String nodeStateText =  bundle.getString(nodeStateKey);

System.out.println("node state: " + nodeStateText);
```

nodeDescription

You can get the administrator assigned description for the UDDI node using the getNodeDescription method of UddiNodeProxy:

1. Invoke getNodeDescription and output:

```
String nodeDescription = uddiNode.getNodeDescription();
System.out.println("node description: " + nodeDescription);
```

nodeApplicationName

The nodeApplicationName attribute is useful for discovering where the UDDI application that corresponds to the UDDI node is installed. The value will be a concatenation of the cell, node and server names, separated by colons. Retrieve the application location using the getApplicationId method of UddiNodeProxy:

1. Invoke getApplicationId and output:

```
String nodeApplicationId = uddiNode.getApplicationId();

System.out.println("node application location: " +
                                        nodeApplicationId);
```

activateNode

Changes the state of the UDDI node to activated, if the UDDI node was previously deactivated.

1. . Invoke activateNode:

```
uddiNode.activateNode();
```

deactivateNode

Changes the state of the UDDI node to deactivated, if the UDDI node was previously activated.

1. Invoke deactivateNode:

```
uddiNode.deactivateNode();
```

initNode

Causes UDDI node initialization, and when this completes the state of the UDDI node is 'activated'.

1. Invoke initNode:

```
uddiNode.initNode();
```

Managing Configuration Properties

UDDI node runtime behavior is affected by the setting of several configuration properties. The UddiNode MBean provides operations to inspect and update their values, as follows: getProperties, getProperty, updateProperty and updateProperties.

See ManagePropertiesSample class for sample code that demonstrates the operations described in this section.

getProperties

Returns collection of all configuration properties as ConfigurationProperty objects.

1. Invoke getProperties:

```
List properties = uddiNode.getProperties();
```

2. Cast each collection member to ConfigurationProperty:

```
if (properties != null) {
  for (Iterator iter = properties.iterator(); iter.hasNext();) {
       ConfigurationProperty property =
                         (ConfigurationProperty) iter.next();
       System.out.println(property);
  }
}
```

Once you have the ConfigurationProperty objects you can inspect attributes like the ID, value, type, whether the property is read only, required for initialization, and get name and description message keys. For example, invoking the toString method returns results similar to:

```
ConfigurationProperty
id: operatorNodeIDValue
nameKey: property.name.operatorNodeIDValue
descriptionKey: property.desc.operatorNodeIDValue
type: java.lang.String
value: uddi:capnscarlet:capnscarlet:server1:default
unitsKey:
```

```
readOnly: true
required: true
usingMessageKeys: false
validValues: none
```

You can use the nameKey and descriptionKey values to look up the translated name and description for a given locale, using the messages.properties resource in the sample package.

getProperty

Returns ConfigurationProperty object with the specified ID. Available property IDs are specified in PropertyConstants together with descriptions of the purpose of the corresponding properties.

1. Invoke getProperty:

```
ConfigurationProperty property =
uddiNode.getProperty(PropertyConstants.DATABASE_MAX_RESULT_COUNT);
```

2. To retrieve the value of the property you could use the getValue method which returns an Object, but in this case, the property is of type integer, so it's easier to retrieve the value using the convenience method getIntegerValue:

```
int maxResults = property.getIntegerValue();
```

updateProperty

Updates the value of the ConfigurationProperty object with the specified ID. Available property IDs are specified in PropertyConstants together with descriptions of the purpose of the corresponding properties. Although you can invoke the setter methods in a ConfigurationProperty object, the only value that is updated in the UDDI node is the value. So to update a property, the steps are typically:

1. Create a ConfigurationProperty object and set its ID:

```
ConfigurationProperty defaultLanguage = new ConfigurationProperty();
defaultLanguage.setId(PropertyConstants.DEFAULT_LANGUAGE);
```

2. Set the value:

```
defaultLanguage.setStringValue("ja");
```

3. Invoke updateProperty:

```
uddiNode.updateProperty(defaultLanguage);
```

updateProperties

Updates several ConfigurationProperty objects in a single request. Set up the ConfigurationProperty objects as for the updateProperty operation.

1. Add updated properties to a List:

```
List updatedProperties = new ArrayList();

updatedProperties.add(updatedProperty1);
updatedProperties.add(updatedProperty2);
```

2. Invoke updateProperties:

```
uddiNode.updateProperties(updatedProperties);
```

Managing Policies

Policies affecting behavior of the UDDI API are managed using the following UddiNode operations: getPolicyGroups, getPolicyGroup, getPolicy, updatePolicy and updatePolicies.

See ManagePoliciesSample class for sample code that demonstrates the attributes and operations described in this section.

getPolicyGroups

Returns collection of all policy groups as PolicyGroup objects.

1. Invoke getPolicyGroups:

```
List policyGroups = uddiNode.getPolicyGroups();
```

2. Cast each collection member to PolicyGroup:

```
if (policyGroups != null) {
for (Iterator iter = policyGroups.iterator(); iter.hasNext();) {
  PolicyGroup policyGroup = (PolicyGroup) iter.next();
  System.out.println(policyGroup);
}
}
```

Each policy group has an ID, name and description key, which you can look up in the messages.properties resource in the sample package. Although the PolicyGroup class does have a getPolicies method, PolicyGroup objects that are returned by the getPolicyGroups operation do not contain any Policy objects. Because of this behavior, clients can determine the known policy groups, and their IDs, without retrieving the entire set of policies in one request. To retrieve the policies within a policy group, use the getPolicyGroup operation.

getPolicyGroup

Returns the PolicyGroup object with the supplied ID.

1. Convert policy group ID to a String:

```
String groupId = Integer.toString(PolicyConstants.REG_APIS_GROUP);
```

2. Invoke getPolicyGroup:

```
PolicyGroup policyGroup = uddiNode.getPolicyGroup(groupId);
```

getPolicy

Returns the Policy object for the specified ID. Like a ConfigurationProperty, a Policy object has an ID, name and description keys, type, value and indicators specifying if the policy is read only or required for node initialization.

1. Convert policy ID to a String:

```
String policyId = Integer.toString(
            PolicyConstants.REG_AUTHORIZATION_FOR_INQUIRY_API);
```

2. Invoke getPolicy:

```
Policy policy = uddiNode.getPolicy(policyId);
```

updatePolicy

Updates the value of the Policy object with the specified ID. Available policy IDs are specified in PolicyConstants together with descriptions of the purpose of the corresponding policies. Although you can invoke the setter methods in a Policy object, the only value that is updated in the UDDI node is the value. So to update a policy, the steps are typically:

1. Create a Policy object and set its ID:

```
Policy updatedPolicy = new Policy();
String policyId =
      Integer.toString(PolicyConstants.REG_SUPPORTS_UUID_KEYS);
updatedPolicy.setId(policyId);
```

2. Set the value:

```
updatedPolicy.setBooleanValue(true);
```

3. Invoke updatePolicy:

```
uddiNode.updatePolicy(updatedPolicy);
```

updatePolicies

Updates several Policy objects in a single request. Set up the Policy objects as for the updatePolicy operation.

1. Add updated policies to a List:

```
List updatedPolicies = new ArrayList();

updatedPolicies.add(updatedPolicy1);
updatedPolicies.add(updatedPolicy2);
```

2. Invoke updatePolicies:

```
uddiNode.updatePolicies(updatedPolicies);
```

Managing Tiers

Tiers control how many of each type of UDDI entities a publisher can save in the UDDI registry. A tier has an ID, an administrator defined name and description, and a set of limits, one for each type of entity. Tiers are managed using the following UddiNode operations: createTier, getTierDetail, getTierInfos, getLimitInfos, setDefaultTier, updateTier, deleteTier and getUserCount.

See ManageTiersSample class for sample code that demonstrates the attributes and operations described in this section.

createTier

Creates a new tier, with specified publish limits for each UDDI entity.

1. Set tier name and description in a TierInfo object.

```
String tierName = "Tier 100";
String tierDescription = "A tier with all limits set to 100.";

TierInfo tierInfo = new TierInfo(null, tierName, tierDescription);
```

2. Define Limit objects for each UDDI entity:

```
List limits = new ArrayList();

Limit businessLimit = new Limit();
businessLimit.setIntegerValue(100);

businessLimit.setId(LimitConstants.BUSINESS_LIMIT);

Limit serviceLimit = new Limit();
serviceLimit.setIntegerValue(100);
serviceLimit.setId(LimitConstants.SERVICE_LIMIT);

Limit bindingLimit = new Limit();
bindingLimit.setIntegerValue(100);
bindingLimit.setId(LimitConstants.BINDING_LIMIT);

Limit tModelLimit = new Limit();
tModelLimit.setIntegerValue(100);
tModelLimit.setId(LimitConstants.TMODEL_LIMIT);

Limit assertionLimit = new Limit();
assertionLimit.setIntegerValue(100);

assertionLimit.setId(LimitConstants.ASSERTION_LIMIT);
limits.add(businessLimit);
limits.add(serviceLimit);
limits.add(bindingLimit);
limits.add(tModelLimit);
limits.add(assertionLimit);
```

3. Create Tier object:

```
Tier tier = new Tier(tierInfo, limits);
```

4. Invoke create Tier and retrieve created tier:

```
Tier createdTier = uddiNode.createTier(tier);
```

5. Inspect generated tier ID of created tier:

```
tierId = createdTier.getId();
System.out.println("created tier has ID: " + tierId);
```

getTierDetail

Returns the Tier object for the given tier ID. The Tier class has getter methods for the tier ID, tier name and description (as set by the administrator), and the collection of Limit objects which specify how many of each UDDI entity type may be published by UDDI publishers allocated to the tier. The isDefault method indicates whether the tier is the default tier, that is, the tier that is allocated to UDDI publishers when auto registration is enabled.

1. Invoke getTierDetail:

```
Tier tier = uddiNode.getTierDetail("2");
```

updateTier

Updates tier contents with the supplied Tier object.

1. Update an existing Tier object (which may have been newly instantiated, or returned by the getTierDetail or createTier operations). This example retains the tier name and description, and all the limit values except the limit being updated:

```
modifiedTier.setName(tier.getName());
modifiedTier.setDescription(tier.getDescription());

Limit tModelLimit = new Limit();
tModelLimit.setId(LimitConstants.TMODEL_LIMIT);
tModelLimit.setIntegerValue(50);

List updatedLimits = new ArrayList();
updatedLimits.add(tModelLimit);

modifiedTier.setLimits(updatedLimits);
```

2. Invoke updateTier:

```
uddiNode.updateTier(modifiedTier);
```

getTierInfos

Returns collection of lightweight tier descriptor objects (TierInfo) which contain the tier ID, and tier name and description values, and whether the tier is the default tier.

1. Invoke getTierInfos:

```
List tierInfos = uddiNode.getTierInfos();
```

2. Output content of each TierInfo:

```
if (tierInfos != null) {

  for (Iterator iter = tierInfos.iterator(); iter.hasNext();) {
     TierInfo tierInfo = (TierInfo) iter.next();
     System.out.println(tierInfo);
  }
}
```

setDefaultTier

Specifies the tier with the given tier ID is the default tier. The default tier is the tier that is allocated to UDDI publishers when auto registration is enabled. Typically this would be set to a tier with low publish limits to prevent casual users publishing too many entities.

1. Invoke setDefaultTier:

```
uddiNode.setDefaultTier("4");
```

deleteTier

Removes the tier with the given tier ID. Tiers can only be removed if they have no UDDI publishers assigned to them, and the tier is not the default tier.

1. Invoke deleteTier:

```
uddiNode.deleteTier("4");
```

getUserCount

Returns the number of UDDI publishers assigned to tier specified by the tier ID.

1. Invoke getUserCount:

```
Integer userCount = uddiNode.getUserCount("4");
System.out.println("users in tier 4: " + userCount.intValue());
```

getLimitInfos

Returns collection of Limit objects representing the limit values for each type of UDDI entity. Limits are used in Tier objects.

1. Invoke getLimitInfos:

```
List limits = uddiNode.getLimitInfos();
```

2. Output the ID and limit value for each Limit object:

```
for (Iterator iter = limits.iterator(); iter.hasNext();) {
  Limit limit = (Limit) iter.next();

  System.out.println("limit ID: "
                     + limit.getId()
                     + ", limit value: "
                     + limit.getIntegerValue());
}
```

Managing UDDI Publishers

UDDI publishers are managed using the UddiNode MBean operations createUddiUser, createUddiUsers, updateUddiUser, deleteUddiUser, getUddiUser, getUserInfos, getEntitlementInfos, assignTier, getUserTier. An example is provided for each, making use of the UddiNodeProxy client class.

See ManagePublishersSample class for sample code that demonstrates the attributes and operations described in this section.

createUddiUser

Registers a single UDDI publisher, in a specified tier, with specified entitlements. The UddiUser class represents the UDDI publisher, and this is constructed using a user ID, a TierInfo object which specifies the tier ID to allocate the UDDI publisher to, and a collection of Entitlement objects which specify what the UDDI publisher is permitted to do.

Tip: to allocate the UDDI publisher default entitlements, set the entitlements parameter to null.

1. Create the UddiUser object:

```
UddiUser user = new UddiUser("user1", new TierInfo("3"), null);
```

2. Invoke createUddiUser:

```
uddiNode.createUddiUser(user);
```

createUddiUsers

Registers multiple UDDI publishers. This example shows how to register 7 UDDI publishers in one call, with default entitlements.

1. Create TierInfo objects for tiers that publishers will be allocated to:

```
TierInfo tier1 = new TierInfo("1");
TierInfo tier4 = new TierInfo("4");
```

2. Create UddiUser objects for each UDDI publisher, specifying tier to allocate to:

```
UddiUser publisher1 = new UddiUser("Publisher1", tier4, null);
UddiUser publisher2 = new UddiUser("Publisher2", tier4, null);
UddiUser publisher3 = new UddiUser("Publisher3", tier4, null);
UddiUser publisher4 = new UddiUser("Publisher4", tier1, null);
UddiUser publisher5 = new UddiUser("Publisher5", tier1, null);
UddiUser cts1 = new UddiUser("cts1", tier4, null);
UddiUser cts2 = new UddiUser("cts2", tier4, null);
```

3. Add the UddiUser objects to a List:

```
List uddiUsers = new ArrayList();

uddiUsers.add(publisher1);
uddiUsers.add(publisher2);
uddiUsers.add(publisher3);
uddiUsers.add(publisher4);
uddiUsers.add(publisher5);
uddiUsers.add(cts1);
uddiUsers.add(cts2);
```

4. Invoke createUddiUsers:

```
uddiNode.createUddiUsers(uddiUsers);
```

updateUddiUser

Updates a UDDI publisher with the details in the supplied UddiUser object. This is typically used to change the tier of one UDDI publisher or update their entitlements. Tip: only supply the entitlements you want to update – the remainder of available entitlements will retain their existing value.

1. Create Entitlement objects with appropriate permission. (the entitlement IDs are found in EntitlementConstants:

```
Entitlement publishUuiDKeyGenerator =
    new Entitlement(PUBLISH_UUID_KEY_GENERATOR, true);
Entitlement publishWithUuidKey =
    new Entitlement(PUBLISH_WITH_UUID_KEY, true);
```

2. Add Entitlement objects to a List:

```
List entitlements = new ArrayList();
entitlements.add(publishUuiDKeyGenerator);
entitlements.add(publishWithUuidKey);
```

3. Update a UddiUser object with the updated entitlements:

```
user.setEntitlements(entitlements);
```

4. Invoke updateUddiUser:

```
uddiNode.updateUddiUser(user);
```

getUddiUser

Retrieves details about a UDDI publisher in the form of a UddiUser object. This specifies the UDDI publisher ID, information about the tier they are assigned to and the entitlements they possess.

1. Invoke getUddiUser:

```
UddiUser user1 = uddiNode.getUddiUser("user1");
```

2. Output the contents of UddiUser:

```
System.out.println("retrieved user: " + user1);
```

getUserInfos

Returns a collection of UserInfo objects. Each UserInfo represents a UDDI publisher known to the UDDI node, and the name of the tier they are allocated to. To get details about a specific UDDI publisher, including the tier ID, and entitlements, use the getUddiUser operation.

1. Invoke getUserInfos:

```
List registeredUsers = uddiNode.getUserInfos();
```

2. Output the UserInfo objects:

```
System.out.println("retrieved registered users: ");
System.out.println(registeredUsers);
```

getEntitlementInfos

Returns a collection of Entitlement objects. Each entitlement is a property that controls whether permission is granted to a UDDI publisher to perform a specified action.

1. Invoke getEntitlementInfos:

```
List entitlementInfos = uddiNode.getEntitlementInfos();
```

2. Specify where to find message resources:

```
String messages = "com.ibm.uddi.v3.management.messages";
ResourceBundle bundle = ResourceBundle.getBundle(
                                    messages, Locale.ENGLISH);
```

3. Iterate through the Entitlement objects, displaying the ID, name and description:

```
for (Iterator iter = entitlementInfos.iterator(); iter.hasNext();) {
  Entitlement entitlement = (Entitlement) iter.next();

  StringBuffer entitlementOutput = new StringBuffer();

  String entitlementId = entitlement.getId();
  String entitlementName =
              bundle.getString(entitlement.getNameKey());
String entitlementDescription =
              bundle.getString(entitlement.getDescriptionKey());

  entitlementOutput.append("Entitlement id: ");
  entitlementOutput.append(entitlementId);
  entitlementOutput.append("\n  name: ");
  entitlementOutput.append(entitlementName);
  entitlementOutput.append("\n  description: ");
  entitlementOutput.append(entitlementDescription);

  System.out.println(entitlementOutput.toString());
}
```

deleteUddiUser

Removes the UDDI publisher with the specified user name (ID) from the UDDI registry.

1. Invoke deleteUddiUser:

```
uddiNode.deleteUddiUser("user1");
```

assignTier

Assigns UDDI publishers with supplied IDs to the specified tier. This is useful when you want to restrict several UDDI publishers, perhaps by assigning them all to a tier that doesn't allow publishing of any entities.

1. Create list of publisher IDs:

```
List uddiUserIds = new ArrayList();

uddiUserIds.add("Publisher1");
uddiUserIds.add("Publisher2");
uddiUserIds.add("Publisher3");
uddiUserIds.add("Publisher4");
uddiUserIds.add("Publisher5");
uddiUserIds.add("cts1");
uddiUserIds.add("cts2");
```

2. Invoke assignTier:

```
uddiNode.assignTier(uddiUserIds, "0");
```

getUserTier

Returns information about the tier a UDDI publisher is assigned to. The returned TierInfo has getters methods for retrieving the tier ID, tier name, tier description, and whether the tier is the default tier.

1. Invoke getUserTier:

```
TierInfo tierInfo = getUserTier("Publisher3");
```

2. Output the contents of the TierInfo object:

```
System.out.println(tierInfo);
```

Managing Value Sets

Value sets are represented in a UDDI registry as value set tModels, with a UDDI types keyedReference with value 'categorization'. Such value sets are backed with a set of valid values and for user defined value sets, this data is loaded into the UDDI registry using UddiNode MBean operations (although it is more convenient to use the User defined value set tool for this purpose). Each value set can be controlled by policy as being supported or not supported. When a value set is supported by policy, it can be referenced within UDDI publish requests. The UddiNode operations available to manage value sets and their data are: getValueSets, getValueSetDetail, getValueSetProperty, updateValueSet, updateValueSets, loadValueSet, changeValueSetTModelKey, unloadValueSet and isExistingValueSet.

See ManageValueSetsSample class for sample code that demonstrates the attributes and operations described in this section.

getValueSets

Returns collection of ValueSetStatus objects.

1. Invoke getValueSets:

```
List valueSets = uddiNode.getValueSets();
```

2. Cast each element to ValueSetStatus and output contents:

```
for (Iterator iter = valueSets.iterator(); iter.hasNext();) {

  ValueSetStatus valueSetStatus = (ValueSetStatus) iter.next();
  System.out.println(valueSetStatus);
}
```

getValueSetDetail

Returns ValueSetStatus object for the given value set tModel key.

1. Invoke getValueSetDetail:

```
uddiNode.getValueSetDetail(
                "uddi:uddi.org:ubr:categorization:naics:2002");
```

2. Retrieve and display details:

```
String name = valueSetStatus.getName();
String displayName = valueSetStatus.getDisplayName();
boolean supported = valueSetStatus.isSupported();

System.out.println("name: " + name);
System.out.println("display name: " + displayName);
System.out.println("supported: " + supported);
```

3. Display value set properties:

```
List properties = valueSetStatus.getProperties();

for (Iterator iter = properties.iterator(); iter.hasNext();) {

  ValueSetProperty property = (ValueSetProperty) iter.next();
  System.out.println(property);
}
```

getValueSetProperty

Returns a property of a value set as a ValueSetProperty object. This is mainly for use by the administrative console to render properties of a value set as a row in a table. For example, one such property is the keyedReference which indicates whether the value set is checked.

1. Invoke getValueSetProperty:

```
uddiNode.getValueSetProperty(
                "uddi:uddi.org:ubr:categorization:naics:2002",
                ValueSetPropertyConstants.VS_CHECKED);
```

2. Read and display boolean value of the property:

```
boolean checked = valueSetProperty.getBooleanValue();

System.out.println("checked: " + checked);
```

updateValueSet

Updates value set status. Only the supported attribute can be updated (all other setter methods are used by the UDDI application).

1. Create a ValueSetStatus object specifying the tModel key and the updated supported value:

```
ValueSetStatus updatedStatus = new ValueSetStatus();
updatedStatus.setTModelKey(
                "uddi:uddi.org:ubr:categorization:naics:2002");
updatedStatus.setSupported(true);
```

2. Invoke updateValueSet:

```
uddiNode.updateValueSet(updatedStatus);
```

updateValueSets

Updates value set status for multiple value sets. As for the updateValueSet operation, only the supported attribute is updated.

1. Populate List with updated ValueSetStatus objects:

```
List valueSets = new ArrayList();

ValueSetStatus valueSetStatus = new ValueSetStatus();
valueSetStatus.setTModelKey(
                "uddi:uddi.org:ubr:categorization:naics:2002");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);
```

```
valueSetStatus = new ValueSetStatus();
 valueSetStatus.setTModelKey(
                "uddi:uddi.org:ubr:categorizationgroup:wgs84");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);

valueSetStatus = new ValueSetStatus();
valueSetStatus.setTModelKey(
                "uddi:uddi.org:ubr:identifier:iso6523:icd");
valueSetStatus.setSupported(false);
valueSets.add(valueSetStatus);
```

2. Invoke updateValueSets:

```
uddiNode.updateValueSets(valueSets);
```

loadValueSet

Loads values for a value set from a UDDI registry V3/V2 taxonomy data file on the local file system. Note: there is also a loadValueSet operation that takes a ValueSetData object but this is only for use by the user defined value set tool.

1. Invoke loadValueSet:

```
uddiNode.loadValueSet(
                      "/valuesets/myvalueset.txt",
                      "uddi:cell:node:server:myValueSet");
```

changeValueSetTModelKey

Any value set values that were allocated to one value set tModel are allocated to the new value set tModel.

1. Invoke changeValueSetTModelKey with old and new tModel keys:

```
uddiNode.changeValueSetTModelKey(
             "uddi:cell:node:server:myValueSet",
             "uddi:cell:node:server:myNewValueSet");
```

unloadValueSet

Unloads values for a value set with the given tModel key.

1. Invoke unloadValueSet:

```
uddiNode.unloadValueSet("uddi:myValueSet");
```

isExistingValueSet

Determines if value set data exists for the given tModel key.

1. Invoke isExistingValueSet and display result:

```
boolean exists = uddiNode.isExistingValueSet(
                "uddi:uddi.org:ubr:categorization:naics:2002");
System.out.println("NAICS 2002 is a value set: " + exists);
```

User-defined value set support in the UDDI registry

The UDDI Version 3 registry provides the structure and modeling tools to find information within a registry effectively. Users can define multiple value sets and add custom value sets. In UDDI Version 2, this functionality was called custom taxonomy support.

Data is worthless if it is lost within a mass of other data and cannot be distinguished or discovered. If a client of UDDI cannot effectively find information within a registry, the purpose of UDDI is considerably compromised. Providing the structure and modeling tools to address this problem is at the heart of UDDI's design. The verification of data within UDDI is core to its mission of description, discovery and integration. It achieves this by several means.

It allows users to define multiple value sets that can be used in UDDI. In such a way, multiple classification schemes can be overlaid on a single UDDI entity. This capability allows organizations to extend the set of such systems UDDI registries support. One is not tied to a single system, but can rather employ several different classification systems simultaneously.

While default value sets are shipped with the product, the UDDI Version 3 registry provides tools enabling 'custom' value sets to be added, potentially enabling UDDI entities to be more specifically categorized when published and further enhancing the capability of client to find specific data.

These value sets can be either checked or unchecked, and this is indicated via a keyedReference in the categoryBag of the tModel that represents a value set (a ″categorization tModel″). These keyedReferences have the tModel key for uddi-org:types and are added to the categoryBag to further describe the behavior of the categorization tModel, as follows:

checked
: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that has a validation service to check that category values are present in a specified value set.

unchecked
: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that does not have a validation service.

The procedure defined below describes how to add additional user-defined value sets, and display their allowed values in the UDDI user console value set tree display. Rational Application Developer has a Web Services Explorer user interface that also allows addition and display of custom checked value sets. The publisher of a value set categorization tModel may specify a 'display name' for use in UDDI user console implementations.

Procedure for adding a user defined value set

To add a user defined value set to the UDDI registry, perform the following tasks:

1. Publish a categorization tModel
2. Load the user defined value set data
3. Set the value set to **supported** status using the Administrative console. To do this you must be a user in an administrative role. This means that user defined value sets cannot be added to the UDDI registry without administrator permission.

The checked value set will only be referenced when the above tasks are complete. Value set data must be provided for validating checked value sets.

Value set data may also be used by user consoles for unchecked value sets, but it is not a requirement and is usually only used for presentation of deprecated value sets, such as unspc-org:unspc and back-level compatibility.

If the value set is checked, any publish requests that have a categoryBag containing keyedReferences with the new categorization tModel will be validated. If there is value set data corresponding to the categorization tModel in the registry database, only valid values will be accepted. If there is no value set data in the database **all** values will be rejected, and the publish request will fail. If the categorization tModel is unchecked, all values will be allowed, regardless of whether there is a corresponding value set present in the UDDI registry database. The value set tModel is not available for use until the administrator enables support for it using the administrative console, or the JMX interface.

Suggested approach

To introduce a new value set:

1. Publish the categorization tModel with a keyedReference of type 'uddi-org:categorization:types' with a key value of **categorization**, a keyedReference of type 'uddi-org:categorization:types' with a Key

Name of **'Checked value set'** and a Key Value of **'checked'**, or a Key Name of **'Unchecked value set'** and a Key Value of **'unchecked'** and a keyedReference of type 'uddi-org:categorization:general_keywords' supplying the value set display name (as described below).
2. Load user defined value set data into the UDDI registry database using the UDDIUserDefinedValueSet utility (described below).
3. Use the administrative Console to set the status of the value set to supported (as described in Value set settings). This can also be achieved directly using the JMX interface.

Note: The SOAP and EJB interfaces will be able to make use of categorization tModels as soon as they are published. However, the UDDI registry user console will require a restart of the UDDI application because it currently gathers its list of categorizations for use in the value set tree display when the application starts.

Publishing a Checked Categorization tModel

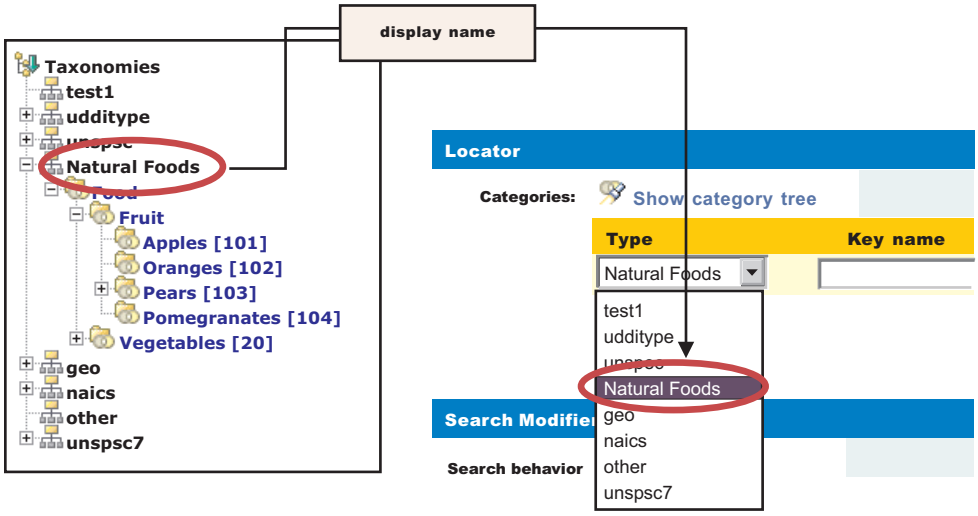
This section describes how to publish a checked categorization tModel with the **'Checked value set'** Key Name for use by a user defined value set.

Publish a tModel to the UDDI registry with a categoryBag containing keyedReferences as follows:

Note	tModelKey	KeyName	KeyValue
1	uddi:uddi-org:categorization:types In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **UDDI Types**	**categorization**	**categorization**
2	uddi:uddi-org:categorization:types In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **UDDI Types**	**Checked value set**	**checked**
3	uddi:uddi-org:categorization:general_keywords In the UDDI registry user interface this tModelKey can be chosen by selecting the category type of **categorization:general_keywords**	**urn:x-ibm:uddi:customTaxonomy:displayName**	*<User Defined Value Set displayName>*

1. Indicates this tModel is a categorization tModel (required).
2. Indicates use of the tModel will be checked against a list of valid data (required). (Omitting this keyedReference, or explicitly specifying a value of 'unchecked' will indicate this categorization is unchecked).
3. Indicates special use of the general keywords value set, with a proprietary uniform resource name (URN) as the keyName value, defines a name for the user-defined value set that is intended for use in user console implementations where the full tModel name might be too long. The value can be 1-255 characters (inclusive) long.

The displayName is intended to provide a way to label a value set such that, when the UDDI user console displays it in a value set tree or in a pull-down list of available value sets, the meaning is clear to the user without being restricted to 8 characters and without needing to be the same as the published tModelName, which could be as long as 255 characters. An example is shown below:



The urn:x-ibm:customTaxonomy:displayName should be unique if only to avoid confusion when displayed in user interfaces but this is not validated.

To publish a new categorization tModel using SOAP, the message would be:

```
<save_tModel generic="3.0" xmlns="urn:uddi-org:api_v3">
  <authInfo></authInfo>>
  <tModel tModelKey="">
    <name>Natural Foods tModel</name>
    <categoryBag>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types" keyName="categorization"
                      keyValue="categorization"/>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types" keyName="Checked value set"
                      keyValue="checked"/>
      <keyedReference tModelKey="uddi:uddi.org:categorization:general_keywords"
                      keyName="urn:x-ibm:uddi:customTaxonomy:displayName" keyValue="Natural Foods"/>
    </categoryBag>
  </tModel>
</save_tModel>
```

Note: to specify an unchecked categorization substitute the key name **'Checked value set'** with **'Unchecked value set'** and **'checked'** Key Value with **'unchecked'** or, more simply, omit the keyedReference completely.

Loading User Defined Value Set Data

User Defined Value Set Data File Format

Value set data is identified by a unique code value, an optional description and a parent code that specifies its relationship with other code values. Value set data must adhere to this format:

Column name	Maximum length	Description of use
Code	765	Unique value within the value set used for validation
description	765	Typically used by UDDI user consoles and optionally in the keyedReference as the keyName value
parentcode	765	Indicates which existing **code** is the logical parent of this one, and is used in tree displays

Typically columns are delimited in the value set data file by '#' characters as in this example:

```
00#Food#00
10#Fruit#00
101#Apples#10
102#Oranges#10
103#Pears#10
1031#Anjou#103
1032#Conference#103
1033#Bosc#103
104#Pomegranates#10
20#Vegetables#00
201#Carrots#20
202#Potatoes#20
203#Peas#20
204#Sprouts#20
```

In the example, 'Food' is the description for the root node with child nodes of 'Fruit' and 'Vegetables' (both of these have parentcode values the same as the code value for 'Food').

The value set data in the example file could then be rendered in a tree like this:

```
Food
  Fruit
    Apples
    Oranges
    Pears
      Anjou
      Conference
      Bosc
    Pomegranates
  Vegetables
    Carrots
    Potatoes
    Peas
    Sprouts
```

The file must be saved in UTF-8 format.

Custom Taxonomy files used in UDDI Version 2 are also supported by the utility.

UDDIUserDefinedValueSet

A utility is provided to load value set data into the UDDI registry, assign existing value set data to another tModel and unload existing value set data. This utility uses the UDDI registry's JMX interface and therefore requires a number of connection parameters.

```
Usage: UDDIUserDefinedValueSet '{'function'}' [options]

function:
-load <path> <key>         Load value set data from specified file
-newKey <oldKey> <newKey>  Move value set to a new tModel
-unload <key>              Unload existing value set

options:
-properties <path>         Specify location of configuration file
-host <host name>          Application Server host
-port <port>               SOAP Lister port number
-node <node name>          Node running a UDDI server
-server <server name>      Server with UDDI deployed
-columnDelimiter <delim>   Character delimiter to denote field end
-stringDelimiter <delim>   Character delimiter to denote strings


Connector security parameters

-userName <name>
-password <password>
```

```
-trustStore <path>
-trustStorePassword <password>
-keyStore <name>
-keyStorePassword <password>
```

Note: Ensure that the command window from which the UDDIUserDefinedValueSet is run is using a suitable codepage and font for displaying the characters contained in the value set name. Use of an incorrect codepage/font may result in unclear messages on a successful load, and create difficulty using the -unload and -newKey options.

The UDDIUserDefinedValueSet script is located in the *app_server_root*/bin directory.

If no connection parameters are supplied a connection is sought on the local host using the default Application Server SOAP port number.

Command arguments are case insensitive.

Usage examples

Load a value set data for a tModel on the local UDDI registry using the percent sign as a column marker in the valuesetdata.txt file.

Move value set data from one checked tModel to another on a UDDI registry in a network deployment configuration.

Unload a value set from a tModel from a server with security turned on supplying the connection and security parameters in myproperties.properties file, but supplying the server and password arguments on the command line (which augment or override those contained in the properties file).

The configuration file, if specified by the optional **-properties** parameter, determines a number of optional parameters. These parameters can be specified on the command line and, if so, override the values in the properties file. These parameters are largely JMX connection parameters and security parameters.

The string.delimiter is typically used where a description value contains the same character as the column delimiter character. For example, if the column.delimiter was set to ',' (a comma), and there was a value set description value of 'Fruits, citrus', you could include this in the value set data file by setting the string.delimiter to ″ (double quote) and enclosing the description in quotes: 'Fruits, citrus'. Note that the quote character is escaped with a backslash ('\') to indicate the literal character is to be used.

If an attempt is made to load a value set to a tModel that has existing value set data, a warning message is given. To override this error provide the **-override** argument. This argument is also required if moving value set data to a new tModel using **-newKey** where the tModel is **checked**, and also unloaded value set data for a **checked** tModel.

Command line arguments and example data	Property and example data	Comments
-columnDelimiter #	column.delimiter=#	The column delimiter that is used in value set data files
-stringDelimiter \″	string.delimiter=\″	The field delimiter (this value must be different to the column.delimiter value)
-host ibm.com®	host=ibm.com	The host name of the system that is running the application server
-port 8880	port=8880	The SOAP port number of the application server

-node ibmNode	node=ibmNode	The name of the node that is running the server with the UDDI registry
-server server1	server=server1	The server that is running the UDDI registry
-userName ibmuser	security.username=ibmuser	The user name. This value is required if WebSphere Application Server security is turned on
-password mypassword	security.password=mypassword	The password
-trustStore /TrustStoreLocation	security.truststore=/TrustStoreLocation	The truststore file location
-keyStore ibmkeystore	security.keystore=ibmkeystore	The keystore name
-trustStorepassword trustpass	security.truststore.password=trustpass	The truststore password
-keyStorePassword keypass	security.keystore.password=keypass	The keystore password

Set the value set to supported

Use the administrative console to set the value set to **supported** by:

- Click *UDDI Nodes* > *<node>* and *Value Sets* (under Additional Properties on the right of the screen)
- Select the Value Set (by checking the box next to it)
- Click *Enable Support* above the list of Value Sets

Validation and Error Handling

The UDDI registry user console performs validation while a save tModel request is being built, that is, before the publish occurs. For example, if the user tries to add two customTaxonomy:displayName keyedReferences the following message is displayed:

```
Advice: Only one 'urn:x-ibm:uddi:customTaxonomy:displayName' key name is allowed for the 'Other' taxonomy.
```

If a keyedReference containing a keyName value that starts with 'urn:x-ibm:uddi:customTaxonomy:' is followed by anything other than 'displayName', the following message is displayed:

```
Advice: Only key name values of 'urn:x-ibm:uddi:customTaxonomy:displayName' are supported.
```

For requests where the save_tModel message may have multiple tModels, if any one of the tModels is a categorization tModel and it fails validation, the request fails with a UDDIInvalidValueException (plus additional information explaining the likely cause), and none of the tModels is published. For example:

```
E_invalidValue (20200) A value that was passed in a keyValue attribute did not pass validation.
This applies to checked categorizations, identifiers and other validated code lists. The error
text will clearly indicate the key and value combination that failed validation. Invalid 'custo
mTaxonomy:dbKey' keyValue [naics] in keyedReference. KeyValue already in use by tModelKey[UUID:
C0B9FE13-179F-413D-8A5B-5004DB8E5BB2]
```

UDDI Utility Tools

The UDDI Utility Tools is a suite of functions that you can use to migrate, move, or copy UDDI Version 2 entities, including child entities and their respective Version 2 entity keys, into a Version 3 UDDI registry.

Note: The UDDI Version 3 publish API supports publisher assigned keys (the Version 2 API did not) and promotion of entities between Version 3 registries can be achieved using normal API functions. UDDI Utility Tools supplied in this release is functionally equivalent to the version supplied in WebSphere Application Server 5.1. However, it is important to know that all UDDI Utility Tools functions in this release are performed using the UDDI Version 2 API. You can export from Version 2 and 3 registries (supplying only the Version 2 representation of the UDDI Entity key) and import into the Version 3 registry, using Version 2 API types. Entities from a Version 3 registry are exported

as Version 2 entities and, as such, elements such as digital signatures will not be present. See section Saving Version 3 entities with a supplied key for an example on how to use the Version 3 API to assign your own keys to Version 3 entities.

Other uses of the tool include:

- Search and select entities from a source UDDI registry by specifying Version 2 keys or search criteria
- Publishing canonical tModels in a UDDI registry, including child entities
- Persist UDDI (Version 2) entities in an intermediate XML representation that can be used to customize and copy those entities to multiple target UDDI Registries, by specifying Version 2 Keys
- Update existing entities in a target UDDI registry, including child entities
- Delete selected entities from a target UDDI registry by specifying Version 2 keys

Use the UDDI Utility Tools by running the UDDIUtilityTools.jar file. This file is located in the *app_server_root*/UDDIReg/scripts directory. Alternatively, you can invoke all of the functions of UDDI Utility Tools through the supplied public Java API.

There are five main functions in UDDI Utility Tools:

Export

Given an entity type and key, or a list of entity types and keys, UDDI Utility Tools gets the UDDI entities from the specified registry and writes them to the UDDI Entity Definition File. The entity type for each key can be one of business, service, bindingTemplate or tModel. The Entity Definition File contains XML that exactly describes each of the specified entities, according to the UDDI Utility Tools schema (which includes the UDDI Version 2 schema). The UDDI Entity Definition File separates entities by type, and automatically detects and records tModels referenced by the specified entities. You can use the 'referenced tModels' section of the file to ensure a target registry includes any referenced tModels before you try to import new entities to that registry.

Import

Given a list of UDDI entities (which can be supplied using the UDDI Entity Definition File generated by the export function, possibly with additional editing, or programmatically in a container object), the import function detects if the entities already exist in the target registry and, if they do not, creates a minimal entity (″stub″) with the specified key. The entities are then published updating the stubs with the supplied data and overwriting, or ignoring, existing entities as specified by the user. Note that the original key is maintained throughout.

Promote

Combines the export and import steps such that the specified entities are extracted (by key) from the source registry and then imported into the target registry in a single logical step. The generation of a UDDI Entity Definition File is optional for this function.

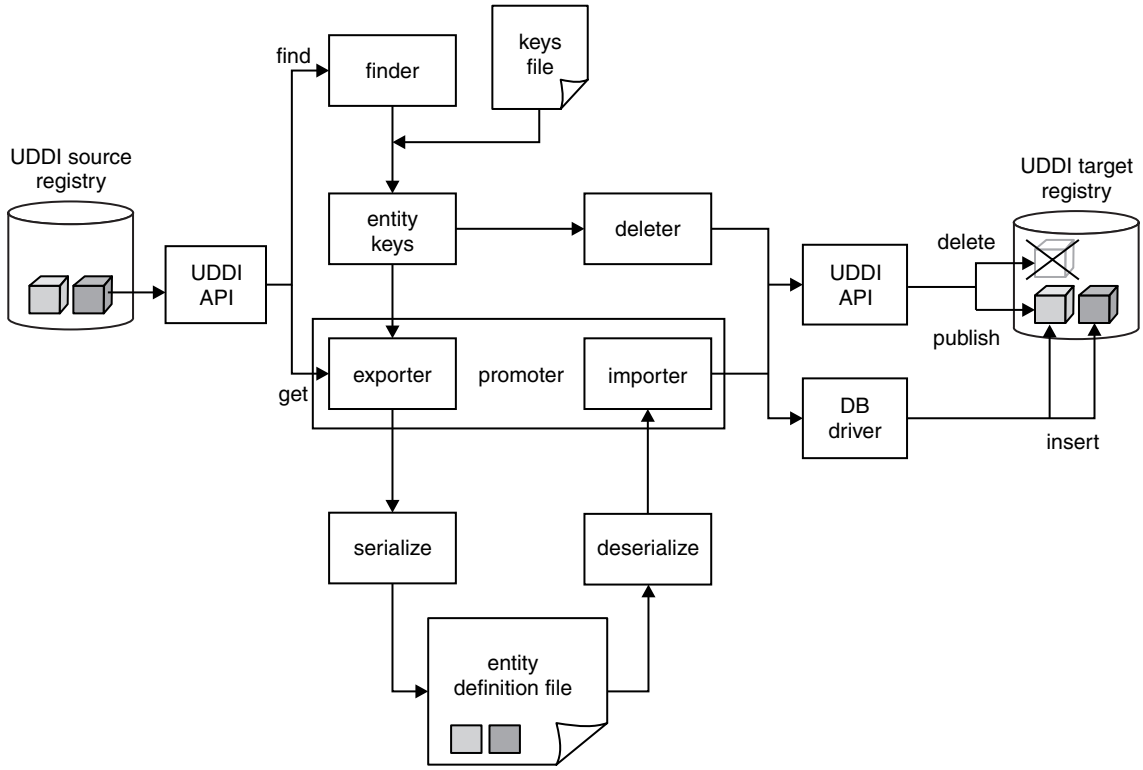
Delete Deletes the specified entities from the target UDDI registry. The entities to delete are specified as an entity type, or a list of entity types, and keys, in the same way as for the export function.

Find Matching Entities

Takes as input search criteria in the form of UDDI Inquiry API objects for each of the various entity types. The set of entities that match the search criteria are used to generate a list of entity keys, and this in turn can be used as input to the export, promote and delete functions.

Note: This function is available only through the programmatic API.

The relationship between the functions, their input and output, and the source and target UDDI Registries is shown in this conceptual overview diagram:



Setting up the configuration file

Configuration data for UDDI Utility Tools resides in a configuration properties file, which describes the runtime environment, UDDI and database locations and access information, logging information, security configuration, entity definition file location, and other flags to control whether referenced entities are to be imported and/or overwritten.

UDDI Utility Tools is distributed with a sample configuration properties file (UDDIUtilityTools.properties) and this is searched for by default in the current directory if no properties path is specified. By default, this file is located in the *app_server_root*/UDDIReg/scripts directory. Modify the file, according to the following list, and specify this modified file when running the utility tools.

- Set the classpath, which should include the current directory (.) and the UDDIUtilityTools.jar itself, plus all the dependent jars, which are listed in the Prerequisites section later on in this topic. The classpath must include the database driver jar (for example db2java.zip).

  If you are configuring a JSSE provider, add the .jar file which contains the provider to the classpath. The configuration of a JSSE provider is optional and is performed by setting the `jsse.provider` property. The default value is com.ibm.jsse.IBMJSSEProvider. To specify the FIPS JSSE provider set the value of the `jsse.provider` property to com.ibm.fips.jsse.IBMJSSEFIPSProvider.
- Set other properties, which are commented in the sample UDDIUtilityTools.properties file as shown below.
- Change localhost to the name of your server.
- Change the port number 9080 to your internal HTTP port.

```
##########################################
# Runtime environment                    #
# (if invoking via java -jar...)         #
# "X Y" required around paths with spaces.   #
# Replace WAS_HOME with your WAS home path.  #
#                                        #
```

```
# db2java.zip is for DB2 - replace this with #
# appropriate database driver file.          #
##############################################
classpath=.:WAS_HOME/UDDIReg/scripts/UDDIUtilityTools.jar:WAS_HOME/plugins/com.ibm.ws.runtime.jar:
WAS_HOME/plugins/com.ibm.uddi.jar:WAS_HOME/lib/j2ee.jar:/QIBM/UserData/Java400/ext/db2_classes.jar

##############################################
# SOAP entry points for source UDDI          #
##############################################
fromInquiryURL=http://localhost:9080/uddisoap/inquiryapi
fromGetURL=http://localhost:9080/uddisoap/get

##############################################
# SOAP entry points for target UDDI          #
##############################################
toInquiryURL=http://localhost:9080/uddisoap/inquiryapi
toPublishURL=http://localhost:9080/uddisoap/publishapi

##############################################
# UDDI Registry user information             #
#                                            #
# Note: this must match the user information #
# that was used to publish the entities on   #
# the target UDDI registry.                  #
##############################################
userID=UNAUTHENTICATED
password=NONE

##############################################
# Configuration for destination UDDI DB      #
# Userid and Password must have authority to #
# the iSeries server and DB                  #
##############################################
dbDriver=com.ibm.db2.jdbc.app.DB2Driver
dbUrl=jdbc:db2:localhost/ibmudi30
dbUser=iSeriesUserProfile
dbPasswd=iSeriesUserPassword

##############################################
# Security provider configuration            #
##############################################
# Indicates whether security is required on the target registry
secure.connection=true

# The location of the truststore if security is required
trustStore.fileName=TrustFile.jks

# The password for the trust store
trustStore.password=WebAS

# The JSSE Provider class name
jsse.provider=com.ibm.jsse.IBMJSSEProvider

##############################################
# Trace and message logging configuration    #
##############################################
# detail level of message output (all functions)
verbose=true

# detail level of trace output.
# 1: severe
# 2: normal
# 3: detail
traceLevel=3

# path to message log file (relative or absolute)
messageLogFileName=logs/messages.log
```

```
# path to trace log file (relative or absolute)
traceLogFileName=logs/trace.log

##########################################
# Miscellaneous Options                  #
##########################################
# indicates if existing entities are overwritten (import/promote)
# Note: tModels in referencedTModels section are never overwritten,
#       regardless of this setting. To overwrite tModels, they must
#       be present in the tModels section.
overwrite=false

# indicates if referenced entities will be imported (import/promote)
importReferencedEntities=true

# location of entity definition file, used for (export/import)
UddiEntityDefinitionFile=definitions/entities01.xml

# namespace prefix to use in definition file (export)
namespacePrefix=promote
```

Prerequisites

Ensure that the following .jar files are available to the UDDI Utility Tools. The locations of the .jar files should be specified in the classpath property in the UDDI Utility Tools properties file:

UDDIUtilityTools.jar
This is the tools JAR itself and is located in *app_server_root*/UDDIReg/scripts.

com.ibm.uddi.jar
This file contains the UDDI4J classes and is located in *app_server_root*/plugins.

j2ee.jar
This file contains some required Java platform for enterprise applications classes, and is located in *app_server_root*/lib.

com.ibm.ws.runtime.jar
This is the Apache SOAP implementation and is located in *app_server_root*/plugins.

DbDriver
This is the driver needed to allow the UDDIUtilityTool to connect to your target database. See the table below for the values you need to specify for your chosen database:

	DB2	Apache Derby	Oracle
DBDriverLocation for classpath on distributed, Windows and i5/OS platforms	*DB2_HOME*/db2java.zip	*app_server_root*/derby/lib/derbyclient.jar	*ORACLE_HOME*/jdbc/lib/ojdbc6.jar
Driver on distributed, Windows and i5/OS platforms	com.ibm.db2.jdbc.app.DB2Driver, or com.ibm.db2.jcc.DB2Driver if you are using a remote DB2 database (you can also set up a local alias to the remote database using the DB2 client)	com.ibm.db2.jcc.DB2Driver	oracle.jdbc.OracleDriver
URL on distributed, Windows and i5/OS platforms	jdbc:db2://host:*database_name*	jdbc:db2j:net://host:1527/*database_name* (see note below)	jdbc:oracle:thin:@host:1521:*database_name*

where

- *app_server_root* is the directory location of WebSphere Application Server.
- *DB2_HOME* is the directory location of DB2, for example c:\Program Files\SQLLIB\java12\
- *ORACLE_HOME* is the directory location of Oracle, for example c:\oracle\ora92\

- *database_name* is the name of the database. For Apache Derby, make sure that *database_name* includes the path to the database, for example *profile_root*/databases/com.ibm.uddi/UDDI30

Note:

- If you are using Apache Derby, make the database network enabled so that it can handle multiple connections. For further details, refer to the section about managing the Derby Network Server in the Derby Server and Administration Guide.
- If you are using DB2, add DB2_HOME/sqllib/lib to your LD_LIBRARY_PATH and LIBPATH environment variables.

The Security provider configuration section in the above properties file shows the location of the default DummyClientTrustFile.jks file. If you are using your own truststore, ensure that the location is placed here.

The UDDI Utility Tools use UDDI Version 2 SOAP inquiry and publish interfaces. These APIs are protected as described in Access control for UDDI registry interfaces. The UDDI Utility Tools also access the UDDI registry database through the database driver, and access to the database is controlled by the database management system.

The UDDI Entity Definition File

You generate this file by the export and promote functions, or you can choose to create it (either by hand, or by modifying a version of the file output by UDDI Utility Tools specifying the export function). It is the input to the import function.

Note: The extension to the uddi:tModel type to add a 'deleted' attribute is not currently used in UDDI Utility Tools.

The file is validated for well formedness and that it complies with the UDDI Utility Tools schema, shown here.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema id="uddiPromote" attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools" xmlns:xsd="http://www.w3.org
                  /2001/XMLSchema"
xmlns:uddi="urn:uddi-org:api_v2" xmlns="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools"
xmlns:promote="http://www.ibm.com/xmlns/prod/WebSphere/UDDIUtilityTools">

 <xsd:import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="xml.xsd" />
 <xsd:import namespace="urn:uddi-org:api_v2" schemaLocation="uddi_v2.xsd" />

 <!-- define a type to represent state of a tModel -->
 <xsd:simpleType name="tModelDeleted">
  <xsd:restriction base="xsd:NMTOKEN">
   <xsd:enumeration value="true" />
   <xsd:enumeration value="false" />
  </xsd:restriction>
 </xsd:simpleType>

 <!-- extend tModel with additional attribute of type tModelDeleted -->
 <!-- This is restricted to values true or false -->
 <xsd:complexType name="tModel">
  <xsd:complexContent>
   <xsd:extension base="uddi:tModel">
    <xsd:attribute name="deleted" type="promote:tModelDeleted" use="optional" />
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>


 <!-- Top level element definitions -->
 <xsd:element name="uddiEntities" type="promote:uddiEntities" />
```

```
<xsd:complexType name="uddiEntities">
 <xsd:sequence>
  <xsd:element ref="promote:tModels" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:businesses" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:services" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:bindings" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="promote:referencedTModels" minOccurs="0" maxOccurs="1" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="businesses" type="promote:businesses" />
<xsd:complexType name="businesses">
 <xsd:sequence>
  <xsd:element ref="uddi:businessEntity" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="tModels" type="promote:tModels" />
<xsd:complexType name="tModels">
 <xsd:sequence>
  <xsd:element ref="uddi:tModel" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="services" type="promote:services" />
<xsd:complexType name="services">
 <xsd:sequence>
  <xsd:element ref="uddi:businessService" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="bindings" type="promote:bindings" />
<xsd:complexType name="bindings">
 <xsd:sequence>
  <xsd:element ref="uddi:bindingTemplate" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>

<xsd:element name="referencedTModels" type="promote:referencedTModels" />
<xsd:complexType name="referencedTModels">
 <xsd:sequence>
  <xsd:element ref="uddi:tModel" minOccurs="0" maxOccurs="unbounded" />
 </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

UDDI Entity Definition File example for canonical tModels

The example Entity Definition File following shows the five main sections for tModels, businesses, services, bindings and referencedTModels:

UDDI Utility Tools can be used to create new UDDI entities in a target UDDI registry. A typical example of this is to introduce a new canonical tModel, which has a publicly known tModel key.

```
<?xml version="1.0" encoding="UTF-8"?>
<promote:uddiEntities xmlns="urn:uddi-org:api_v2" xmlns:promote="http://www.ibm.com/xmlns/prod/WebSphere/
                      UDDIUtilityTools">

  <!-- tModels -->
  <promote:tModels>

    <tModel tModelKey="uuid:ee3966a8-faa5-416e-9772-128554343571" >
      <name>http://schemas.xmlsoap.org/ws/2002/07/policytmodel</name>
      <description>WS-PolicyAttachment policy expression</description>
    </tModel>
```

```
    <tModel tModelKey="uuid:ad61de98-4db8-31b2-a299-a2373dc97212" >
      <name>uddi-org:wsdl:address</name>
 <description xml:lang="en">
This tModel is used to specify the URL fact that the address must be obtained from the WSDL deployment
file.
      </description>
      <overviewDoc>
        <overviewURL>
http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm#Address
        </overviewURL>
      </overviewDoc>
    </tModel>

  </promote:tModels>

  <!-- businesses -->
  <promote:businesses>
  </promote:businesses>

  <!-- services -->
  <promote:services>
  </promote:services>

  <!-- bindings -->
  <promote:bindings>
  </promote:bindings>

  <!-- referenced tModels -->
  <promote:referencedTModels>
  </promote:referencedTModels>

</promote:uddiEntities>
```

Starting UDDI Utility Tools at a command prompt

Ensure that you are using the correct level of Java code by setting the PATH statement to include the Java code that is supplied with WebSphere Application Server. For example, from the command line, type:

UDDI Utility Tools can be started using:

java - jar UDDIUtilityTools.jar <function> [options]
: using a specified properties file that sets up classpath and other parameters, or it can be called using:

java CommandLineProcessor
: where CommandLineProcessor is the class which processes command line arguments for UDDI Utility Tools, sets up configuration and invokes the appropriate function.

Note: Before you run UDDIUtilityTools.jar from the command line, ensure that you have edited the UDDIUtilityTools.properties file. If you have saved this properties file in a different directory from the directory containing the UDDIUtilityTools.jar file, make sure you specify the location of the properties file as part of the command line arguments. See the Setting up the configuration file section earlier in this topic for more details.

The usage is as follows:

```
Usage: java -jar UDDIUtilityTools.jar {function} [options]

function:
  -promote <entity source>   Promote entities between registries
  -export <entity source>    Extract entities from registry to XML
  -delete <entity source>    Delete entities from registry
  -import                    Create entities from XML to registry

where <entity source> is one of:
  -tmodel|-business|-service|-binding <key> Specify single entity type and key
```

```
  -keysFile | -f <filename>  Specify file containing entity types and keys

options:
  -properties <filename>     Specify path to configuration file
  -overwrite | -o            Overwrite an entity if it already exists
  -log | -v                  Output verbose messages
  -definitionFile <filename> Specify path to UDDI entity definition file
  -importReferenced          Import entities referenced by source entities

The following options override property settings in configuration file:
  -overwrite
  -log
  -definitionFile
  -importReferenced

Example: java -jar UDDIUtilityTools.jar -promote -keysFile /uddikeys.txt
```

Below are a set of UDDI Utility Tools command line examples:

To export a single business to the EDF file specified in a properties file in the current directory.

```
java -jar UDDIUtilityTools.jar -export -business 28B8B928-2B2E-4EC9-A647-1E40651E4752
```

As above but this time using a keys file to specify the entities to be exported

```
java -jar UDDIUtilityTools.jar -export -keysFile /myKeyFiles/keyFile01.txt
```

As above but also specifying verbose output to appear on the command line.

```
java -jar UDDIUtilityTools.jar -export -keysFile /myKeyFiles/keyFile02.txt -v
```

To import the contents of the default EDF specified in a UDDIUtilitiyTools.properties file in the current directory.

```
java -jar UDDIUtilityTools.jar -import
```

As above but also specifying that referenced tModels should be imported into the target registry.

```
java -jar UDDIUtilityTools.jar -import -importReferenced
```

To import the entities from an EDF at the specified location.

```
java -jar UDDIUtilityTools.jar -import -definitionFile /myEDFs/entities01.xml
```

To import the entities from the default EDF including referenced tModels. Overwrite specifies that any entities excluding referenced tModels that are found in the target registry should be overwritten.

```
java -jar UDDIUtilityTools.jar -import -overwrite -importReferenced
```

To promote a single service from a source to a target registry using the properties file at a specified location.

```
java -jar UDDIUtilityTools.jar -promote -service 67961D67-330F-4F14-8210-E74A58E710F3
-properties /UUT/myUUTProps.properties
```

To promote a set of entities specified in a keys file.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile03.txt
```

As above but specifying that existing entities in the target registry get overwritten.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile04.txt -overwrite
```

To promote a set of entities specified in a keys file including referenced tModels.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile05.txt -importReferenced
```

To promote a set of entities specified in a keys file but also create an EDF containing the promoted entities.

```
java -jar UDDIUtilityTools.jar -promote -keysFile /myKeyFiles/keyFile06.txt
-definitionFile /myEDFs/entities02.xml
```

To logically delete a single tModel. Note that it is not possible to physically delete tModels.

```
java -jar UDDIUtilityTools.jar -delete -tModel UUID:1E2B9D1E-E53D-4D36-9D46-6CCC176C466A
```

To delete all the entities specified in the keys file. Note that with the exception tModels all other entities will be physically deleted from the target registry.

```
java -jar UDDIUtilityTools.jar -delete -keysFile /myKeyFiles/keyFile04.txt
```

A keys file example

The following example shows the keys that are to be exported, promoted, or deleted from the target registry:

```
#
# Keys of entities to be exported, promoted from source registry or deleted from target registry
#
# Note: keys must be comma separated and on SAME line
# Note: property names are case sensitive. ('tmodels=' will be ignored)

businesses=97C77097-AC6C-4CA0-A6C4-452F7045C470, 4975E949-581F-4FCA-AD5F-E08280E05F9F
services=BB3864BB-1578-4833-8179-14391F14791F
bindings=
tModels=273F1727-7BFF-4FB5-A1FD-BA5C45BAFD9C
```

Note: If the importReferenced property is set to true, the list of tModels in the referencedTModels section is imported to the target registry. Minimal entities are created if the referencedTModel is new. If the referencedTModel already exists it is never overwritten, regardless of the overwrite property value. This is so that commonly referenced tModels such as categorization tModels do not keep being updated unnecessarily.

Should you need to update a referencedTModel, you must manually move the referencedTModel definition to the tModels section in the entity definition file and set overwrite to true.

Content of the log files

The following examples show the contents of two of the log files that are produced by running the tool. Some comments have been added in square brackets and in italic to highlight important points in the log file. The first is the messages.log, which shows successful and unsuccessful operations for export, import and delete functions:

```
[29/07/04 17:39:57:531 BST] CWUDU0002I: ********** Starting UDDI Utility Tools ********** [timestamp and
 eyecatcher indicate when tool is run]
[29/07/04 17:39:57:531 BST] CWUDU0009I: Exporting entities...
[29/07/04 17:39:57:531 BST] CWUDU0015I: Exported 14 entities.
[29/07/04 17:39:57:531 BST] CWUDU0029I: Serializing...
[29/07/04 17:39:57:531 BST] CWUDU0030I: Serialized entities.
[29/07/04 17:39:57:531 BST] CWUDU0016I: Importing entities...
[29/07/04 17:39:57:531 BST] CWUDU0124I: Created tModel minimal entity with tModelKey [uuid:667e2766-4781-
4151-b3a0-809f7180a096].
[29/07/04 17:39:57:531 BST] CWUDU0121I: Created business minimal entity with businessKey [263f5526-8708-4
834-9f5d-8f8c878f5d6e].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [0af2a30a-be70-401
f-a027-331a6c332712].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [61012761-d02c-4c7
0-ae98-435ffd4398f9].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [f97af9f9
-7cb7-47bd-8b90-b55e4db590df].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [17e4c017
```

```
-d273-43ec-af4a-f9b841f94a30].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [9e2c239e
-3b30-40a9-9c25-ce64edce25b9].
[29/07/04 17:39:57:531 BST] CWUDU0121I: Created business minimal entity with businessKey [49bb6949-4b0e-4
e81-88a7-e26bfbe2a7f1].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [003d2b00-f6c0-407
1-8b84-f235a2f28445].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [df1019df
-2d2f-4f32-bf18-4f21274f1835].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [b229aeb2
-f2b1-4115-a06f-536753536f10].
[29/07/04 17:39:57:531 BST] CWUDU0122I: Created service minimal entity with serviceKey [84d8e584-2510-409
9-9b2a-6023f1602a0a].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [62a9a762
-7fff-4f7a-8463-af0c79af63ee].
[29/07/04 17:39:57:531 BST] CWUDU0123I: Created binding template minimal entity with bindingKey [e08654e0
-b212-42c0-bcf3-655e9765f392].
[29/07/04 17:39:57:531 BST] CWUDU0115I: Imported 7 entities and 0 referenced entities. [this kind of
message indicates the operation worked!]
[29/07/04 17:39:57:531 BST] CWUDU0002I: ********** Starting UDDI Utility Tools **********
[29/07/04 17:39:57:531 BST] CWUDU0023I: Deleting entities...
[29/07/04 17:39:57:531 BST] CWUDU0028I: Deleted 7 entities.
```

The second log file shows a typical trace log file entry for an export:

```
[29/07/04 17:39:57:531 BST] ********** Starting UDDI Utility Tools ********** [eyecatcher and timestamp
indicate when tool is run]
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()  [the '>' indicates
entry to the constructor of this class]
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded tModel keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded business keys
TransformConfiguration:
  nameSpacePrefix=promote
  uddiEntityDefinitionFile=/temp/MigToolFiles/Results/Promote_api_EDF_1.xml

ExportConfiguration:
  fromGetURL=http://yottskry:9080/uddisoap/
  fromInquiryURL=http://yottskry:9080/uddisoap/inquiryAPI

ImportConfiguration:
  overwrite=true
  uddiEntityDefinitionFile=/temp/MigToolFiles/Results/Promote_api_EDF_1.xml
  importReferencedEntities=true

PublishConfiguration:
  toInquiryURL=http://davep:9080/uddisoap/inquiryAPI
  toPublishURL=http://yottskry:9080/uddisoap/publishAPI
  userID=Publisher1
  trustStoreFileName=/WebSphere600/AppServer/etc/DummyClientTrustFile.jks
  secureConnection=false

DatabaseConfiguration:
  dbDriver=com.ibm.db2.jcc.DB2Driver
  dbURL=jdbc:db2:LOC1
  dbUser=db2admin

LoggerConfiguration:
  messageStream=null
  messageLogFileName=/temp/MigToolFiles/logs/message.log
  traceLogFileName=/temp/MigToolFiles/logs/trace.log
  traceLevel=3
  verbose=true


[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI()
[29/07/04 17:39:57:531 BST] ********** Starting UDDI Utility Tools **********
```

```
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded tModel keys [ log
entries without a '>' or '<' are status messages only ]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded business keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded service keys
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() loaded binding keys
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UddiEntityKeys()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.UddiEntityKeys() [the '<' indicates exit from the
constructor]
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.export.KeyFileReader() removed duplicate, empty and
null keys
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.export.KeyFileReader()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.PromoterAPI.setUddiEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.PromoterAPI.deleteEntities()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] < com.ibm.uddi.promoter.publish.EntityDeleter()
[29/07/04 17:39:57:531 BST] > com.ibm.uddi.promoter.UDDIClient()
[29/07/04 17:39:57:531 BST]   com.ibm.uddi.promoter.UDDIClient() client type: 1
```

Starting UDDI Utility Tools through the API

UDDI Utility Tools provides a public API to functions for exporting, importing, promoting, finding and deleting UDDI entities. All of these functions can be invoked by using the PromoterAPI class. Usage of this class to perform these functions is typically to:

1. Create a Configuration object and populate it from a Properties object or from a configuration properties file.
2. Create a PromoterAPI object passing the Configuration in the constructor.
3. For keys based functions (export, delete and promote), set the keys by supplying a UDDIEntityKeys object, the location of the keys file, or, for one entity, by specifying an entity type and a key value.
4. Invoke the corresponding method for the function required: exportEntities, promoteEntities(boolean), importEntities, deleteEntities or extractKeysFromInquiry(FindTModel, FindBusiness, FindService, FindBinding, FindRelatedBusinesses).

There is some sample code for UDDI Utility Tools, demonstrating usage of the API classes, available from Samples Central.

The ″low-level″ API classes and methods have been deprecated in this release. Refer to the API documentation for details.

Known limitations with UDDI Utility Tools and workarounds

There are some known limitations with UDDI Utility Tools and a workaround for each. See UDDI troubleshooting tips for more information.

Embedded Apache Derby Restriction

The ’export’ and ’delete’ functions when referencing a source registry with an embedded Apache Derby database are supported. However, the ’import’ and ’promote’ functions are not supported when referencing a target registry because of a limitation with the UDDI registry when working with an embedded Apache Derby database. To allow the ’promote’ and ’import’ functions to work, the embedded Apache Derby database needs to be made network enabled. For information about configuring network Apache Derby, refer to the section about managing the Derby Network Server in the Derby Server and Administration Guide.

Saving Version 3 entities with a supplied key

An example of saving a Version 3 business with a defined key is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <save_business xmlns="urn:uddi-org:api_v3">
      <authInfo>a399c4a3-6387-47cd-a1bd-91f7bb91bdd7</authInfo>
      <businessEntity businessKey="uddi:mycompany-p1.com:computers">
        <name xml:lang="en">WithKey</name>
      </businessEntity>
    </save_business>
  </Body>
</Envelope>
```

Known limitations with UDDI Utility Tools and workarounds

There are known limitations with the UDDI Utility Tools and a workaround for each:

- PublisherAssertions are not supported and will not be promoted.

  Workaround: After the user has promoted the businesses that are related, he must recreate the publisherAssertion relationship.
- Referenced businesses in service projections are not added automatically to the EDF in the same manner as referenced tModels.

  Workaround: Add the referenced business that will 'own' the projected service to the EDF. If the business is not present in the target registry, it should be placed before the service's owning business in the EDF.
- Cycle detection for service projections are not detected in the same manner as for referenced tModels.

  Workaround: If a circular reference is present between two or more service projections, break the cycle by removing one of the projections temporarily, perform the import and update the changed entity to reestablish the cycle in the target registry.
- tModels that were deleted (in the logical sense) in the source registry are imported and promoted as undeleted in the target registry. This is because, in the UDDI Version 2 specification, the deleted state of tModels is not exposed as API calls.

  Workaround: After importing the tModel, perform a delete. This is done using the UDDI Utility Tools delete function, or any other UDDI registry API access method.
- BindingTemplates referenced by hostingRedirectors are not added automatically to the EDF in the same manner as referenced tModels.

  Workaround: Add the referenced bindingTemplate to the EDF.
- Businesses referenced by an 'owningBusiness' keyedReference are not automatically added to the EDF.

  Workaround: Import the referenced business into the target registry before importing the tModel that references it.
- A few combinations of command line arguments are not validated and prevented, for example, it is possible to specify -import with -keysFile <path to file> in the same command, although the -keysFile is ignored.

Java API for XML Registries (JAXR) provider for UDDI

The Java API for XML Registries (JAXR) is a Java client API for accessing both UDDI (Version 2 only) and ebXML registries. It is part of the Java EE specification.

The JAXR API comprises the Java Platform, Enterprise Edition (Java EE) packages javax.xml.registry and javax.xml.registry.infomodel. There is Java EE API documentation at Web Services Reference.

The preferred UDDI Java client APIs are:

- UDDI4J Version 2, for UDDI Version 2
- UDDI Version 3 Client for Java, for UDDI Version 3

JAXR provider

The current JAXR specification (Version 1.0) defines a JAXR provider as an implementation of the JAXR API. Generally, a JAXR provider can be a JAXR provider for UDDI, a JAXR provider for ebXML, or a pluggable provider that supports both UDDI and ebXML. The JAXR provider for UDDI is a provider for UDDI only.

UDDI versions

A JAXR provider for UDDI accesses a UDDI registry using the UDDI Version 2 SOAP APIs only. The UDDI registry for UDDI Version 3 in this version of WebSphere Application Server supports the UDDI Version 1, 2 and 3 SOAP APIs. Therefore you can use the JAXR provider for UDDI to access this registry. You can also use the JAXR provider to access the UDDI registry for UDDI Version 2 in WebSphere Application Server Version 5.x.

To work with the UDDI Version 3 SOAP APIs, use the UDDI Version 3 Client for Java; you cannot use JAXR.

Capability level

The JAXR specification defines two capability profiles, capability level 0 and capability level 1. The JAXR API documentation categorizes each JAXR method as either level 0 or level 1. Generally, a JAXR provider for UDDI has capability level 0 and supports all level 0 methods, while A JAXR provider for ebXML has capability level 1 and supports all level 0 and level 1 methods. The JAXR provider for UDDI is a capability level 0 provider, and supports only level 0 methods.

JAXR for UDDI - getting started and further information

A sample program demonstrates how to get started with the Java API for XML Registries (JAXR). This topic also discusses class libraries, authentication and security, internal taxonomies, and logging and messages.

A simple sample

The following sample program shows how to obtain the ConnectionFactory instance, create a Connection to the registry and save an Organization in the registry.

Replace all references to ″localhost″ with references to the specific iSeries machine on which the UDDI registry is running.

```
import java.net.PasswordAuthentication;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.Properties;
import java.util.Set;

import javax.xml.registry.BulkResponse;
import javax.xml.registry.BusinessLifeCycleManager;
import javax.xml.registry.Connection;
import javax.xml.registry.ConnectionFactory;
import javax.xml.registry.JAXRException;
import javax.xml.registry.RegistryService;
import javax.xml.registry.infomodel.Key;
import javax.xml.registry.infomodel.Organization;

public class JAXRSample
{
    public static void main(String[] args) throws JAXRException
    {
        //Tell the ConnectionFactory to use the JAXR Provider for UDDI
        System.setProperty("javax.xml.registry.ConnectionFactoryClass",
                                                    "com.ibm.xml.registry.uddi.ConnectionFactoryImpl");
```

```
        ConnectionFactory connectionFactory = ConnectionFactory.newInstance();

        //Set the URLs for the UDDI inquiry and publish APIs.
        //These must be the URLs of the UDDI version 2 APIs.
        Properties props = new Properties();
        props.setProperty("javax.xml.registry.queryManagerURL", "http://localhost:9080/uddisoap/inquiryapi");
        props.setProperty("javax.xml.registry.lifeCycleManagerURL", "http://localhost:9080/uddisoap/publishapi");
        connectionFactory.setProperties(props);

        //Create a Connection to the UDDI registry accessible at the above URLs.
        Connection connection = connectionFactory.createConnection();

        //Set the user ID and password used to access the UDDI registry.
        PasswordAuthentication pa = new PasswordAuthentication("Publisher1", new char[] { 'p', 'a', 's',
                                                                                          's', 'w', 'o', 'r', 'd' });
        Set credentials = new HashSet();
        credentials.add(pa);
        connection.setCredentials(credentials);

        //Get the javax.xml.registry.BusinessLifeCycleManager interface, which contains
        //methods corresponding to UDDI publish API calls.
        RegistryService registryService = connection.getRegistryService();
        BusinessLifeCycleManager lifeCycleManager = registryService.getBusinessLifeCycleManager();

        //Create an Organization (UDDI businessEntity) with name "Organization 1".
        Organization org = lifeCycleManager.createOrganization("Organization 1");

        //Add the Organization to a Collection, ready to be saved in the UDDI registry.
        Collection orgs = new ArrayList();
        orgs.add(org);

        //Save the Organization in the UDDI registry.
        BulkResponse bulkResponse = lifeCycleManager.saveOrganizations(orgs);

        //Obtain the Organization's Key (the UDDI businessEntity's businessKey) from the response.
        if (bulkResponse.getExceptions() == null)
        {
            //1 Organization was saved, so 1 key will be returned in the response collection
            Collection responses = bulkResponse.getCollection();
            Key organizationKey = (Key)responses.iterator().next();
            System.out.println("\nOrganization Key = " + organizationKey.getId());
        }
    }
}
```

Classpath

The class libraries of the JAXR Provider for UDDI are contained within the com.ibm.uddi_1.0.0.jar file, located in the *app_server_root*/plugins directory. When using the JAXR API from within a J2EE application running under WebSphere Application Server, all required classes will automatically be on the classpath. When using the JAXR API from outside this environment, the following jars must be on the Java classpath:

- *app_server_root*/lib/bootstrap.jar
- *app_server_root*/lib/j2ee.jar
- *app_server_root*/plugins/com.ibm.uddi_1.0.0.jar
- *app_server_root*/plugins/com.ibm.ws.runtime_6.1.0

javax.xml.registry.ConnectionFactory

To use the JAXR Provider for UDDI, the name of the ConnectionFactory implementation class must first be specified by setting the System Property “javax.xml.registry.ConnectionFactoryClass” to “com.ibm.xml.registry.uddi.ConnectionFactoryImpl”. Failure to specify this will result in the value defaulting to “com.sun.xml.registry,common.ConnectionFactoryImpl”, which will not be found. This will result in a JAXRException when the ConnectionFactory.newInstance() method is called. The JAXR Provider for UDDI does not support lookup of the ConnectionFactory via JNDI.

javax.xml.registry.Connection Properties

Connection specific properties must be specified by setting a java.util.Properties object on the JAXR ConnectionFactory before obtaining a Connection. The JAXR specification defines the full list of these properties. The table below lists the three most important properties, and what values they should take in order to use the JAXR Provider for UDDI to access the UDDI registry. The only required Connection property is “javax.xml.registry.queryManagerURL”, however it is recommended that “javax.xml.registry.lifeCycleManagerURL” is also set, and that the default value of “javax.xml.registry.security.authenticationMethod” is understood. The rest of the Connection properties defined in the JAXR specification are optional, and their values are not specific to the UDDI registry. The JAXR Provider for UDDI does not define any additional provider-specific properties.

Property	Description
javax.xml.registry.queryManagerURL	The URL of the UDDI registry’s inquiry API for UDDI Version 2. Typically this will be of the form: “http://<hostname>:<port>/uddisoap/inquiryapi”. This property is required.
javax.xml.registry.lifeCycleManagerURL	The URL of the UDDI registry’s publish API for UDDI v2. Typically this will be of the form: “http://<hostname>:<port>/uddisoap/publishapi”. If this property is not specified, it defaults to the value of the javax.xml.registry.queryManagerURL property, however the UDDI registry will typically have different URLs for the inquiry and publish APIs, and it is recommended to specifiy both properties.
javax.xml.registry.authenticationMethod	The method of authentication to use when authenticating with the registry. This may take one of two values, “UDDI_GET_AUTHTOKEN” and “HTTP_BASIC”. The default value is “UDDI_GET_AUTHTOKEN” if none is specified. See section Authentication and Security below for more information.

Authentication and security

Authentication

The javax.xml.registry.authenticationMethod Connection property tells the JAXR Provider which method to use when authenticating with the UDDI registry. The two supported values of this property are “UDDI_GET_AUTHTOKEN” and “HTTP_BASIC”. The JAXR Provider for UDDI does not support the “CLIENT_CERTIFICATE” or “MS_PASSPORT” methods of authentication. If this property is not set, the default authentication method is “UDDI_GET_AUTHTOKEN”.

UDDI_GET_AUTHTOKEN

The JAXR Provider uses the UDDI V2 get_authToken API to authenticate with the registry. The get_authToken call is made automatically by the JAXR Provider when the Connection credentials are set, and the UDDI V2 authToken returned by the call is saved by the JAXR Provider for use on subsequent UDDI publish API calls.

HTTP_BASIC

The JAXR Provider uses HTTP basic authentication to authenticate with the registry. This is supported by WebSphere Application Server when security is on. No UDDI V2 get_authToken API call is made, instead the username and password are sent in the HTTP headers using HTTP basic authentication every time a UDDI API call is made (both inquiry and publish). If the UDDI registry does not require HTTP basic authentication, the credentials are ignored.

The JAXR Provider uses UDDI Version 2 SOAP inquiry and publish APIs. These APIs are protected as described in Access control for UDDI registry interfaces.

USING SSL (Secure Sockets Layer)

SSL can be used to encrypt HTTP traffic between the JAXR Provider for UDDI and the UDDI registry. To use SSL, the JAXR client program should do the following:

1. When setting the “javax.xml.registry.queryManagerURL” and “javax.xml.registry.lifeCycleManagerURL” Connection properties, specify a URL with the protocol “https” and the correct port for using SSL to access the UDDI registry. The UDDI registry’s default port for HTTPS is 9443. Often only the lifeCycleManager URL (the UDDI Publish API URL) will require SSL.
2. Add a new Security Provider to the java.security.Security object, according to the JSSE (Java Secure Sockets Extension) implementation being used. If running under the JVM provided in WebSphere Application Server, the JSSE provided by IBM will automatically be on the classpath. Add the IBM Security Provider as follows:

   ```
   java.security.Security.addProvider(new com.ibm.jsse.JSSEProvider());
   ```
3. Set the System property “javax.net.ssl.trustStore” to be the file name of the client trust store file. The client trust store file is a Java key store (.jks) file and must contain the server certificate of the UDDI registry. Key store files can be managed using the ikeyman tool
4. Set the System property “javax.net.ssl.trustStorePassword”. This is the password used to open the client trust store file.
5. If using an IBM version of JVM that is older than the level provided in this version of WebSphere Application Server, it may be necessary to set the System property “java.protocol.handler.pkgs” to “com.ibm.net.ssl.internal.www.protocol”. For more information on SSL and the ikeyman tool refer to SSL and IKEYMAN within this Information Center.

Internal taxonomies

The JAXR Provider for UDDI supplies the following internal taxonomies:

Taxonomy	ClassificationScheme name (UDDI tModel name)	ClassificationScheme id (UDDI Version 2 tModelKey)
NAICS 1997	ntis-gov:naics:1997	UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
NAICS 2002	ntis-gov:naics:2002	UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2
UNSPSC 3.1	unspsc-org:unspsc:3-1	UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384
UNSPSC 7	unspsc-org:unspsc	UUID:CD153257-086A-4237-B336-6BDCBDCC6634
ISO3166 2003	ubr-uddi-org:iso-ch:3166-2003	UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88

The tModels corresponding to all of these taxonomies are available in the UDDI Version 3 registry. If using the JAXR Provider to access a UDDI Version 2 registry, only the tModels corresponding to NAICS 1997, UNSPSC 3.1 and ISO3166 are available.

Custom internal taxonomies

A user may supply their own custom internal taxonomies. To create a new custom internal taxonomy and make it available to the JAXR provider, follow these steps:

1. Create a text file containing the taxonomy element data. As an example, look at the file iso3166-2003-data.txt in plugins/com.ibm.uddi_1.0.0. This is the taxonomy data file for the supplied ISO 3166 taxonomy. The first few lines are:

   ```
   iso3166#--#World#--
   iso3166#AD#Andorra#--
   iso3166#AE#United Arab Emirates#--
   ```

```
iso3166#AE-AJ#'Ajm?n#AE
iso3166#AE-AZ#Ab? Z?aby[Abu Dhabi]#AE
iso3166#AE-DU#Dubayy [Dubai]#AE
iso3166#AE-FU#Al Fujayrah#AE
iso3166#AE-RK#Ra's al Khaymah#AE
iso3166#AE-SH#Ash Sh?riqah [Sharjah]#AE
iso3166#AE-UQ#Umm al Qaywayn#AE
iso3166#AF#Afghanistan#--
iso3166#AF-BAL#Balkh#AF
iso3166#AF-BAM#B?m??n#AF
```

Each line represents one element of the taxonomy, or one Concept in the taxonomy Concept tree. Each line has the form:

```
<taxonomy ID>#<element value>#<element name>#<parent element value>
```

Token	Description
<taxonomy ID>	The taxonomy ID is the same for every element of a taxonomy.
<element value>	The Concept value (UDDI keyValue).
<element name>	The Concept name (UDDI keyName).
<parent element value>	This defines the element's parent element in the taxonomy tree. For every element (except the root element) in the data file, there should be another line which defines the element's parent element. The root element is denoted by defining itself as its own parent. There should be only one root element, and no parentless elements.
#	The delimiter character. This does not have to be "#" and can be defined for each taxonomy in the taxonomyConfig.properties file.

2. Save a ClassificationScheme (UDDI tModel) in the UDDI registry to represent the new internal taxonomy. This can be done using the javax.xml.registry.BusinessLifeCycleManager.saveClassificationSchemes() method.
3. Add the new taxonomy to the taxonomyConfig.properties file:
   a. Copy the supplied taxonomyConfig.properties file from the root of the com.ibm.uddi_1.0.0.jar file.

      The content of the supplied taxonomyConfig.properties file is:

      ```
      naics-1997 = UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2, naics-1997-data.txt, #
      naics-2002 = UUID:1FF729F2-1948-46CF-B660-31EC107F1663, naics-2002-data.txt, #
      unspsc  = UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384, unspsc-data.txt,     #
      unspsc7_data = UUID:CD153257-086A-4237-B336-6BDCBDCC6634, unspsc7-data.txt,    #
      iso3166-2003 = UUID:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88, iso3166-2003-data.txt,#
      ```

      This file has one line per supplied internal taxonomy, which is of the form:

      ```
      <taxonomy ID> = <tModelKey>,<data filename>,<data file delimiter>
      ```

Token	Description
<taxonomy ID>	This is used internally by the JAXR provider to identify each taxonomy. It does not have to be the same as the taxonomy ID in the corresponding taxonomy data file.
<tModelKey>	The tModelKey of the corresponding UDDI tModel. (The id of the corresponding JAXR ClassificationScheme).
<data filename>	The name of the corresponding taxonomy data file.
<data file delimiter>	The delimiter character used in the taxonomy data file. All supplied internal taxonomies use "#", but user-supplied internal taxonomies may use different delimiters.

   b. Add a new line for the new taxonomy to the copy of the taxonomyConfig.properties file. Do not remove any existing taxonomies from the file as this will make them unavailable to the JAXR provider.
4. Add the copied taxonomyConfig.properties file to the Java classpath ahead of jaxruddi.jar.

5. If any JAXR client programs are still running that were started before the new taxonomy was added to the taxonomyConfig.properties file, a new Connection must be created in order to pick up the new taxonomy.

Important notes on internal taxonomies

Each internal taxonomy is loaded into memory once per JAXR Connection. The taxonomy's ClassificationScheme is created when the Connection is created. At this time the associated UDDI tModel is obtained from the registry and used to populate the ClassificationScheme attributes. The taxonomy's Concept object tree is not created until the first time the ClassificationScheme is requested by the user. All subsequent requests for the same internal taxonomy using the same Connection will return the same object tree.

Modification of the Concept object tree

Because there is only one ClassificationScheme and Concept object tree per internal taxonomy per Connection, a user should not attempt to modify programmatically any part of the Concept tree, because all future requests for this taxonomy using the same Connection will return the modified (and now possibly invalid) objects. Programmatic modification of the Concept tree will not result in any changes to the associated taxonomy data file. If a user wishes to make a change to the values in a user-defined internal taxonomy, they must first make the changes in the taxonomy data file, and then create a new Connection to pick up the changes in a new Concept tree.

Modification of the ClassificationScheme

Similarly, a user should not attempt to modify programmatically an internal ClassificationScheme, except in the case where a user wishes to modify and then save a user-defined internal ClassificationScheme. A new Connection is not required to pick up programmatic changes.

Logging and messages

UDDI4J Logging

The JAXR Provider for UDDI uses UDDI4J Version 2 to communicate with the UDDI registry. UDDI4J has its own logging which can be switched on by setting the value of the System property "org.uddi4j.logEnabled" to "true". This outputs to the standard error log the XML request and response bodies of every UDDI request.

Trace

Entry, exit, exception, warning and debug trace is provided using commons-logging. See http://jakarta.apache.org/commons/logging/ for more information on commons-logging. Trace will only be created if the JAXR client configures it. Entry, exit and debug trace uses the debug level of logging. Exception and warning trace uses the info level of logging. Additionally, info level logging is provided before each UDDI4J request is made.

Standard error log messages

The InternalTaxonomyManager, EnumerationManager and PostalSchemeManager send warning messages to System.err if error conditions occur that do not warrant an exception, but that the user should be informed of. Examples of these are if a taxonomy data file contains an invalid line, or if a tModel corresponding to an internal taxonomy could not be found in the registry.

Planning to use Web services

You can plan to develop and implement Web services based on a variety of Java programming models.

Before you begin

The Samples Gallery includes Samples that demonstrate JAX-WS-based Web services. The JAX-WS Web services Samples demonstrate the simple message exchange patterns using both synchronous and asynchronous invocation of Web services in SOAP 1.1 and SOAP 1.2 environments. The Samples are composed with Web service standards such as WS-Addressing (WS-A) , WS-Reliable Messaging (WS-RM), and WS-Secure Conversation (WS-SC), which you can use to complete a broad range of interoperability tests. The samples demonstrate the use of JavaBeans artifacts and static service endpoints and proxy-based clients. Additionally, a Sample is provided that demonstrates Message Transmission Optimization Mechanism (MTOM).

About this task

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

You must re-write existing JAX-RPC applications if you want to take advantage of the features of the JAX-WS programming model.

Web services reflect the service-oriented architecture approach to programming. This approach is based on the idea of building applications by discovering and implementing network-available services, or by invoking the available applications to accomplish a task. Web services deliver interoperability, for example, Web services applications provide a way for components created in different programming languages to work together as if they were created using the same language. Web services rely on existing transport technologies, such as HTTP, and standard data encoding techniques, such as Extensible Markup Language (XML), for invoking the implementation.

1. Identify your goals and design Web services to fit your e-business solution. Consider what you want to accomplish by using Web services. Decide how Web services fit into your current topology, applications and programming model. Determine how the Web services process requests on the server and how the clients manage and use the Web service.
2. Design your Web services for reliability, availability, manageability and security. For example, you want your Web services to process a transaction in a reasonable time at all hours of the day and provide users with optimal security, such as authentication for buyers. Planning to use Web services to work with WebSphere Application Server helps to meet these requirements.
3. Review the standards used in developing and deploying Web services onto WebSphere Application Server. Development and deployment are based on a variety of Java programming models.
4. Decide what development and implementation tools to use. You can use a variety of manual development and implementation tasks. Whether you have an existing Web service to implement or you want to develop your own from a JavaBeans implementation or from an Enterprise JavaBeans (EJB) module, you can choose different tasks respective to your resources. You can also use assembly tools to complete development and implementation tasks.
5. Install the application server. For detailed information on installing the application server, read about installing your application serving environment.
6. Review Web services Samples.

Results

You have a design plan for implementing Web services applications into your business architecture.

Learning about the Web Services Invocation Framework (WSIF)

The Web Services Invocation Framework (WSIF) is a WSDL-oriented Java API. You use this API to invoke Web services dynamically, regardless of the service implementation format (for example enterprise bean) or the service access mechanism (for example Java Message Service (JMS)).

About this task

Using WSIF, you can move away from the usual Web services programming model of working directly with the SOAP APIs, towards a model where you interact with representations of the services. You can therefore work with the same programming model regardless of how the service is implemented and accessed.

To learn about WSIF, see the following topics:

- "Goals of WSIF."
- "WSIF Overview" on page 458.
  1. "WSIF architecture" on page 458.
  2. "WSIF and WSDL" on page 459.
  3. "WSIF usage scenarios" on page 459.

Goals of WSIF

WSIF aims to extend the flexibility provided by SOAP services into a general model for invoking Web services, irrespective of the underlying binding or access protocols.

SOAP bindings for Web services are part of the WSDL specification, therefore when most developers think of using a Web service, they immediately think of assembling a SOAP message and sending it across the network to the service endpoint, using a SOAP client API. For example: using Apache SOAP the client creates and populates a Call object that encapsulates the service endpoint, the identification of the SOAP operation to invoke, the parameters to send, and so on.

While this process works for SOAP, it is limited in its use as a general model for invoking Web services for the following reasons:

- "Web services are more than just SOAP services."
- "Tying client code to a particular protocol implementation is restricting" on page 457.
- "Incorporating new bindings into client code is hard" on page 457.
- "Multiple bindings can be used in flexible ways" on page 457.
- "A freer Web services environment enables intermediaries" on page 457.

The goals of the Web Services Invocation Framework (WSIF) are therefore:

- To give a binding-independent mechanism for Web service invocation.
- To free client code from the complexities of any particular protocol used to access a Web service.
- To enable dynamic selection between multiple bindings to a Web service.
- To help the development of Web service intermediaries.

Web services are more than just SOAP services

You can deploy as a Web service any application that has a WSDL-based description of its functional aspects and access protocols. If you are using the Java Platform, Enterprise Edition (Java EE) environment, then the application is available over multiple transports and protocols.

For example, you can take a database-stored procedure, expose it as a stateless session bean, then deploy it into a SOAP router as a SOAP service. At each stage, the fundamental service is the same. All that changes is the access mechanism: from Java DataBase Connectivity (JDBC) to Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and then to SOAP.

The WSDL specification defines a SOAP binding for Web services, but you can add binding extensions to the WSDL so that, for example, you can offer an enterprise bean as a Web service using RMI-IIOP as the access protocol. You can even treat a single Java class as a Web service, with in-thread Java method invocations as the access protocol. With this broader definition of a Web service, you need a binding-independent mechanism for service invocation.

Tying client code to a particular protocol implementation is restricting

If your client code is tightly bound to a client library for a particular protocol implementation, it can become hard to maintain.

For example, if you move from Apache SOAP to Java Message Service (JMS) or enterprise bean, the process can take a lot of time and effort. To avoid these problems, you need a protocol implementation-independent mechanism for service invocation.

Incorporating new bindings into client code is hard

If you want to make an application that uses a custom protocol work as a Web service, you can add extensibility elements to WSDL to define the new bindings. But in practice, achieving this capability is hard.

For example you have to design the client APIs to use this protocol. If your application uses just the abstract interface of the Web service, you have to write tools to generate the stubs that enable an abstraction layer. These tasks can take a lot of time and effort. What you need is a service invocation mechanism that allows you to update existing bindings, and to add new bindings.

Multiple bindings can be used in flexible ways

To take advantage of Web services that offer multiple bindings, you need a service invocation mechanism that can switch between the available service bindings at run time, without having to generate or recompile a stub.

Imagine that you have successfully deployed an application that uses a Web service which offers multiple bindings. For example, imagine that you have a SOAP binding for the service and a local Java binding that lets you treat the local service implementation (a Java class) as a Web service.

The local Java binding for the service can only be used if the client is deployed in the same environment as the service. In this case, it is more efficient to communicate with the service by making direct Java calls than by using the SOAP binding.

If your clients could switch the actual binding used based on run-time information, they could choose the most efficient available binding for each situation.

A freer Web services environment enables intermediaries

Web services offer application integrators a loosely-coupled paradigm. In such environments, intermediaries can be very powerful.

Intermediaries are applications that intercept the messages that flow between a service requester and a target Web service, and perform some mediating task (for example logging, high-availability or transformation) before passing on the message. The Web Services Invocation Framework (WSIF) is designed to make building intermediaries both possible and simple. Using WSIF, intermediaries can add value to the service invocation without needing transport-specific programming.

WSIF Overview

The Web Services Invocation Framework (WSIF) provides a Java API for invoking Web services, independent of the format of the service or the transport protocol through which it is invoked.

This framework addresses all of the issues identified in “Goals of WSIF” on page 456.

WSIF provides the following features:
- An API that provides binding-independent access to any Web service.
- A close relationship with WSDL, so it can invoke any service that you can describe in WSDL.
- A stubless and completely dynamic invocation of a Web service.
- The capability to plug a new or updated implementation of a binding into WSIF at run time.
- The option to defer the choice of a binding until run time.

WSIF provides runtime support for Web services, and for WSDL extensions and bindings, that were not known at build time. This capability is known as *dynamic invocation*. Using WSIF, a client application can choose dynamically the optimal binding to use for invoking Web service operations. For example, a Web service might offer a SOAP binding, and also a local Java binding so that you can treat the local service implementation (a Java class) as a Web service. If a client application is deployed in the same environment as the service, then this client can use the local Java binding for the service. This provides more efficient communication between the client and the service by making direct Java calls rather than indirect calls using the SOAP binding. WSIF provides this runtime support through the use of providers. The providers support Web services, WSDL extensions and bindings that were not known at build time by using the WSDL description to access the target service.

WSIF is designed to work both in an unmanaged environment (stand-alone) and inside a managed container. You can use the Java Naming and Directory Interface (JNDI) to find the WSIF service, or you can use the location described in the WSDL.

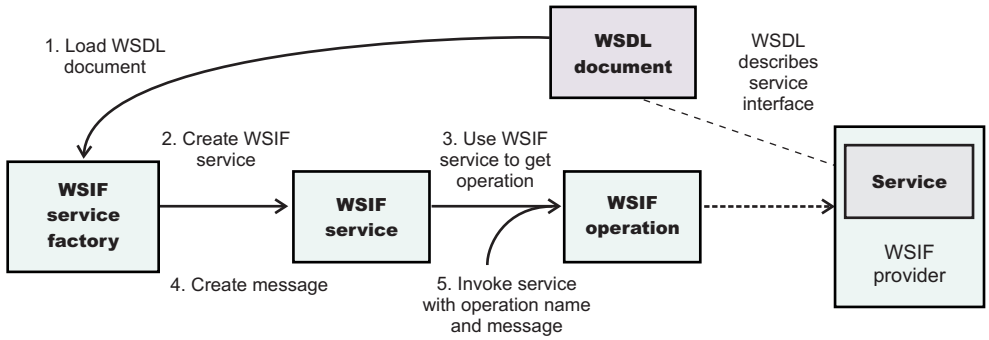
For more conceptual information about WSIF and WSDL, see the following topics:
- WSIF and WSDL
- WSIF architecture
- WSIF usage scenarios

WSIF supports Internet Protocol Version 6, and Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 for SOAP.

WSIF architecture

A diagram depicting the Web Services Invocation Framework (WSIF) architecture, and a description of each of the major components of the architecture.

The Web Services Invocation Framework (WSIF) architecture is shown in the figure.



The components of this architecture include:

WSDL document
The Web service WSDL document contains the location of the Web service. The binding document defines the protocol and format for operations and messages defined by a particular portType.

WSIF service
The WSIFService interface is responsible for generating an instance of the WSIFOperation interface to use for a particular invocation of a service operation. For more information, see the WSIFService interface topic.

WSIF operation
The run-time representation of an operation, called *WSIFOperation* is responsible for invoking a service based on a particular binding. For more information, see the WSIFOperation interface topic.

WSIF provider
A WSIF provider is an implementation of a WSDL binding that can run a WSDL operation through a binding-specific protocol. WSIF includes SOAP providers, JMS providers, Java providers and EJB providers. For more information, see Linking a WSIF service to the underlying implementation of the service.

WSIF and WSDL

There is a close relationship between the metadata-based Web Services Invocation Framework (WSIF) and the evolving semantics of Web Services Description Language (WSDL).

In WSDL, a service is defined in three distinct sections:
- The **portType**. This section defines the abstract interface offered by the service. A portType defines a set of *operations*. Each operation can be In-Out (request-response), In-Only, Out-Only and Out-In (Solicit-Response). Each operation defines the input and/or output *messages*. A message is defined as a set of *parts*, and each part has a schema-defined type.
- The **binding**. This section defines how to map between the abstract portType and a real service format and protocol. For example the SOAP binding defines the encoding style, the SOAPAction header, the namespace of the body (the targetURI), and so on.
- The **port**. This section defines the actual location (endpoint) of the available service. For example, the HTTP Web address at which a SOAP service is available.

Currently in WSDL, each port has one and only one binding, and each binding has a single portType. But (more importantly) each service (portType) can have multiple ports, each of which represents an alternative location and binding for accessing that service.

The Web Services Invocation Framework (WSIF) follows the semantics of WSDL as much as possible:
- The WSIF dynamic invocation API directly exposes run-time equivalents of the model from WSDL. For example, invocation of an operation involves executing an operation with an input message.
- WSDL has extension points that support the addition of new ports and bindings. This enables WSDL to describe new systems. The equivalent concept in WSIF is a provider, that enables WSIF to understand a class of extensions and thereby to support a new service implementation type.

As a metadata-based invocation framework, WSIF follows the design of the metadata. As WSDL is extended, WSIF is updated to follow.

The primary type system of WSIF is XML schema. WSIF supports invocation using dynamic proxies, which in turn support Java type systems, but when you use the WSIFMessage interface it is your responsibility to populate WSIFMessage objects with data based on the XML schema types as defined in the WSDL document. You should define your object types by a canonical and fixed mapping from schema types into the run-time environment.

WSIF usage scenarios

This topic describes two brief scenarios that illustrate the role WSIF plays in the emerging Web services environment.

Scenario: Redevelopment and redeployment

When you first implement a Web service, you create a simple prototype. When you want to move a prototype Web service into production, you often need to redevelop and redeploy it.

The Web Services Invocation Framework (WSIF) uses the same API calls irrespective of the underlying technologies, therefore if you use WSIF:

- You can reimplement and redeploy your services without changing the client code.
- You can use existing reliable and high-performance infrastructures like Remote Method Invocation over Internet Inter-ORB Protocol (RMI-IIOP) and Java Message Service (JMS) without sacrificing the location-independence that the Web service model offers.

Scenario: Service Flow composition

A service flow typically invokes a Web service, then passes the response from one Web service to the next Web service, perhaps performing some transformation in the middle.

There are two key aspects to this flow that WSIF provides:

- A representation of the service invocation based on the metadata in WSDL.
- The ability to build invocations based solely on the portType, which can therefore be used in any implementation.

For example, imagine that you build a meta-service that uses a number of services to build a process. Initially, several of those services are simple Java bean prototypes that are written and exposed through SOAP, but you plan to reimplement some of them as EJB components, and to out-source others.

If you use SOAP, it ties up multiple threads for every onward invocation, because they pass through the Web server and servlet engine and on to the SOAP router. If you use WSIF to call the beans directly, you get much better performance compared to SOAP and you do not lose access or location transparency. Using WSIF, you can replace the Java bean implementations with EJB implementations without changing the client code. To move some of the Web services from local implementations to external SOAP services, you just update the WSDL.

Setting up and deploying a new UDDI registry

A UDDI registry node consists of the UDDI registry application (an enterprise application that is supplied as part of WebSphere Application Server), a store of data (using a relational database management system) referred to as the UDDI database, and a means to connect the application to the data (a datasource and related elements). To set up a new UDDI registry you create the UDDI database and datasource, and deploy the supplied application.

Before you begin

Start WebSphere Application Server, and create a server to host the UDDI registry. Use Starting and stopping quick reference for information about starting WebSphere Application Server using either commands or the administrative console.

About this task

The subtopics describe how to create the UDDI database (which can be local or remote) and datasource, and how to deploy the UDDI registry application.

You can create either a *default* UDDI node or a *customized* UDDI node. The main difference between default and customized, in the context of these set up tasks, refers to a number of mandatory UDDI registry properties such as the UDDI node ID and description, and the prefix to be used for generated discovery URLs.

Default UDDI node
The mandatory properties are automatically set to default values and cannot be changed. A default UDDI node is a suitable option for initial evaluation of the UDDI registry, and for development and test purposes.

Customized UDDI node
You must set the mandatory properties, but once set they cannot be changed for this configuration. With a customized UDDI node you have more control over the database management system used for the UDDI database, and the properties used to set up the UDDI database. With a customized UDDI node, you create the UDDI database and datasource to your own specifications before deploying the UDDI registry application. A customized node is a suitable option for production purposes. To move from a default UDDI node to a customized node, see "Changing the UDDI registry application environment after deployment" on page 483.

- If you want to quickly set up a UDDI registry for test or development purposes, follow the instructions in "Setting up a default UDDI node with a default datasource" on page 462. The database, datasource and UDDI registry application are created or deployed by a single script. Note that the database type is embedded Apache Derby.
- If you want to create a default UDDI registry with a database other than embedded Apache Derby, or if you want an embedded Apache Derby database but you want to create the datasource manually, follow the instructions in "Setting up a default UDDI node" on page 463.
- If you want to create a customized UDDI registry, follow the instructions in "Setting up a customized UDDI node" on page 472.

Database considerations for production use of the UDDI registry

The UDDI registry fully supports a number of databases and can be used for development and test purposes. However, there are factors to consider when you decide which database is appropriate for your anticipated UDDI registry production use.

It is important to consult the information that is supplied by your chosen database vendor, but you also need to consider the likely size and volume of requests, and whether the general performance and scalability of the UDDI registry is important.

For example, the Apache Derby database supports the full function of the UDDI registry, but it is not an enterprise level database and it does not have the same characteristics, for example, scaling or performance, as enterprise databases such as DB2.

Note: Apache Derby is not supported for production use.

If you need multiple connections to the UDDI registry database (for example to use the UDDI registry in a cluster configuration) and Apache Derby is your preferred database, you need to use the network option for Apache Derby. This is because embedded Apache Derby has a limitation that allows only one Java virtual machine to access or load a database instance at any one time. That is, two application servers cannot access the same Apache Derby database instance at the same time.

Note: The UDDI registry can support multiple users, even if a single database connection exists.

Related concepts

"Overview of the Version 3 UDDI registry" on page 410
The Universal Description, Discovery, and Integration (UDDI) specification defines a way to publish and discover information about Web services. The term *Web service* describes specific business functionality that is exposed by a company, usually through an Internet connection, to allow another company, its subsidiaries, or software program, to use the service.

Related reference

"About Apache Derby" on page 1538
Use Apache Derby as a test and development database only. Apache Derby must run at a minimal version of v10.3 or Cloudscape® v10.1x. The Apache Derby package that is bundled with the application server is backed by full IBM Quality Assurance (QA).

"Data source minimum required settings for Apache Derby" on page 1436
These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Apache Derby and Cloudscape data sources.

Setting up a default UDDI node with a default datasource

Use this task to create a UDDI node with predetermined property values and an embedded Apache Derby database.

About this task

You cannot change the mandatory node properties, such as node ID, either during the creation of the node, or afterward. Such a node is suitable for initial evaluation of the UDDI registry and for development and test purposes. If you want to choose the mandatory node properties yourself, set up a customized node as detailed in "Setting up a customized UDDI node" on page 472.

The UDDI database and datasource are automatically created by running a single script, which also deploys the UDDI registry application. If you want to have more control over the datasource, refer to "Setting up a default UDDI node" on page 463.

1. Create the UDDI node by running the wsadmin script uddiDeploy.jacl from the *app_server_root*/bin directory. The syntax of the command is shown below.

```
wsadmin [-conntype none] [-profileName profile_name] -wsadmin_classpath app_server_root/derby/lib
               -f uddiDeploy.jacl
               node_name
               server_name
               default
```

   where

   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile will be used.
   - '-conntype none' is optional, and is only needed if the application server is not running.
   - *app_server_root* is the directory name of the WebSphere Application Server installation location.
   - *node_name* is the name of the WebSphere node on which the target server runs. Note that the node name is case sensitive.
   - *server_name* is the name of the target server on which you wish to deploy the UDDI registry, such as server1. Note the server name entered is case sensitive.
   - 'default' causes the command to create a UDDI node, with default policies, within an Apache Derby database and datasource. This is a special case only for Apache Derby and creates everything required to run a UDDI node.

     If the Apache Derby database already exists, you are asked if you want to recreate it. If you choose to recreate the database, your existing database is deleted and a new one is created in its place. If you choose not to recreate the database, the command exits and a new database is not created.

Note: If the application server has already accessed the existing Apache Derby database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in “Removing a UDDI registry node” on page 978, restart the server, and run the uddiDeploy.jacl script again.

For example, to create a UDDI node called ’MyNode’ on server ’server1’, you might enter the following (this assumes server1 is started):

```
wsadmin -profileName myProfile -wsadmin_classpath /QIBM/ProdData/WebSphere/
AppServer/V61/Base/derby/lib -f uddiDeploy.jacl MyNode server1 default
```

If the server is not started, the command is:

```
wsadmin -conntype none -profileName myProfile -wsadmin_classpath /QIBM/ProdData/
WebSphere/AppServer/V61/Base/derby/lib -f uddiDeploy.jacl MyNode server1 default
```

(Enter these commands as one command on a single line.)

2. Click **Applications** → **Application Types** → **WebSphere enterprise applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively, if the application server is not already running, start the application server. This action automatically starts the UDDI registry application. The UDDI node is now active.

   Note: Restarting the UDDI application, or the application server, always reactivates the UDDI node, even if the node was previously deactivated.

3. Click **UDDI** → **UDDI Nodes** → *UDDI_node_id* to display the properties page for the UDDI registry node. Set **Prefix for generated discoveryURLs** to a valid URL for your configuration. This property specifies the URL prefix that is applied to generated discovery URLs that are used by the HTTP GET service for UDDI Version 2.

What to do next

Follow the instructions in “Using the UDDI registry Installation Verification Program (IVP)” on page 482 to verify that you have successfully set up the UDDI node.

Setting up a default UDDI node

Use this task to create a UDDI node with predetermined property values. This UDDI node is suitable for initial evaluation of the UDDI registry and for development and test purposes.

About this task

You cannot change the mandatory node properties, such as node ID, either during the creation of the node, or afterward. If you want to choose your own mandatory node properties, set up a customized node, as detailed in “Setting up a customized UDDI node” on page 472.

1. Create a database schema to hold the UDDI registry by completing one of the following tasks, ensuring that you use the default node options where specified:
   - “Creating a DB2 distributed database for the UDDI registry” on page 464
   - “Creating a DB2 for iSeries database for the UDDI registry” on page 465
   - “Creating an Apache Derby database for the UDDI registry” on page 467
   - “Creating an Oracle database for the UDDI registry” on page 468
2. Set up a data source for the UDDI registry application to use to access the database, as described in “Creating a data source for the UDDI registry” on page 469.
3. Deploy the UDDI registry application, as described in “Deploying the UDDI registry application” on page 471.
4. Click **Applications** → **Application Types** → **WebSphere enterprise applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively, if the application server is not already running, start the application server. This action automatically starts the UDDI registry application. The UDDI node is now active.

Note: Restarting the UDDI application, or the application server, always reactivates the UDDI node, even if the node was previously deactivated.

5. Click **UDDI** → **UDDI Nodes** → *UDDI_node_id* to display the properties page for the UDDI registry node. Set **Prefix for generated discoveryURLs** to a valid URL for your configuration. This property specifies the URL prefix that is applied to generated discovery URLs that are used by the HTTP GET service for UDDI Version 2.

What to do next

Because you have chosen to use a default UDDI node, the node will be initialized when the UDDI application is started for the first time. Follow the instructions in "Using the UDDI registry Installation Verification Program (IVP)" on page 482 to verify that you have successfully set up the UDDI node.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

<DataBaseName>
The name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

<DB2UserID>
A DB2 userid with administrative privileges.

<DB2Password>
The password for the DB2 userid.

<BufferPoolName>
The name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, because the buffer pool is created as part of this task.

<TableSpaceName>
The name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
The name of a temporary table space. A suggested value is udditstemp, but any name can be used, because the temporary table space is created as part of this task.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt.
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```

where <DataBaseName> is the name of the database being created.

5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`
   e. `force application all`
   f. `terminate`
   g. `stop`
   h. `start`
7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
9. [Optional] Enter the following command if you want the database to be used as a default UDDI node:
   `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for iSeries database for the UDDI registry

Perform this task if you want to use DB2 for iSeries as the database store for your UDDI registry data.

Before you begin

IBMUDI30 and IBMUDS30 are the default names of the UDDI registry schema in the SQL scripts mentioned below. These default names are the recommended values and are assumed throughout the UDDI documentation. If you want to use different names, modify the SQL files listed below, then substitute your own names whenever you see 'IBMUDI30' or 'IBMUDS30' in other sections of the documentation.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Use iSeries Navigator to run SQL scripts.
   a. Open iSeries Navigator.
   b. Expand **My Connections** → **iSeriesName** → **Databases**.
   c. Select **iSeriesName**.
   d. Right-click **Run SQL Scripts...**.

   A **Run SQL Scripts** window opens.
2. Open the i5/OS DB2 SQL files.
   a. Map a network drive to the root directory of your i5/OS server's integrated file system.
   b. In Windows Explorer, expand the `WAS_HOME/UDDIReg/databaseScripts` directory.
   c. Open the following SQL files with a text editor (such as Notepad):
      - uddi30crt_10_prereq_db2_iSeries.sql
      - uddi30crt_20_tables_generic_iSeries.sql
      - uddi30crt_25_tables_db2udb_iSeries.sql
      - uddi30crt_30_constraints_generic_iSeries.sql
      - uddi30crt_35_constraints_db2udb_iSeries.sql
      - uddi30crt_40_views_generic_iSeries.sql
      - uddi30crt_45_views_db2udb_iSeries.sql
      - uddi30crt_50_triggers_db2udb_iSeries.sql
      - uddi30crt_60_insert_initial_static_data_iSeries.sql
3. Copy the text to the **Run SQL Scripts** window.
   a. In the text editor of the file `uddi30crt_10_prereq_db2_iSeries.sql`, click **Edit** → **Select All**.
   b. Click **Edit** → **Copy**.
   c. In the **Run SQL Scripts** window, click **Edit** → **Paste**.
   d. Click **Run** → **All**.
   e. After the script completes running, select all the SQL text and delete it from the **Run SQL Scripts** window.
   f. Repeat the above steps for all the SQL scripts listed in step 2.

   Note: iSeries Navigator for i5/OS V5R2 will encounter an error on the last line of the SQL script when running the `uddi30crt_25_tables_db2udb_iSeries.sql` script. This error can be ignored. i5/OS V5R2 does support running UDDI in a clustered high availability environment. For more information, see the WebSphere Application Server for i5/OS tech notes.
4. This step should only be run if you want your database to be used as a default UDDI node.
   a. Open `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 2 above.
   b. Copy and run `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 3 above.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Apache Derby database for the UDDI registry

Perform this task to use Apache Derby (embedded or network) as the database store (either local or remote) for your UDDI registry.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

arg1 The path of the SQL files. On a standard installation, this is *app_server_root*/UDDIReg/databasescripts

arg2 The path to the location where you want to install the Apache Derby database.

For example, *profile_root*/databases/com.ibm.uddi

arg3 The name of the Apache Derby database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

arg4 An optional argument, which must either be the string 'DEFAULT', or be omitted. Specify DEFAULT if you want the database to be used as a default UDDI node. This argument is case sensitive.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Apache Derby database using UDDIDerbyCreate.jar.

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/jre/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Apache Derby database already exists, you are asked if you want to recreate it. If you choose to recreate the database, your existing database is deleted and a new one is created in its place. If you choose not to recreate the database, the command exits and a new database is not created.

   Note: If the application server has already accessed the existing Apache Derby database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 978, restart the server, and run the uddiDeploy.jacl script again.
3. If you are using a remote database (which requires network Apache Derby), or you want to use network Apache Derby for other reasons, for example, if you want to use Apache Derby with a cluster, configure the Apache Derby Network Server framework. For details, see the section about managing the Derby Network Server in the Derby Server and Administration Guide.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Perform this task if you want to use Oracle as the database store for your UDDI registry data. You need only do this once for each UDDI registry, as part of setting up and deploying a UDDI registry.

About this task

The supported versions of Oracle are Version 9i[1] and Version 10g[2]

You can create a remote database using Oracle, but not a local database. Because the database is remote, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql
```

      - For Version 10g and later:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql
```

   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`

1.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

2.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

2. This last command should only be run if you want the database to be used as a default UDDI node.

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
```

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a data source for the UDDI registry

You create a data source so that the UDDI registry can use it to access the UDDI database.

Before you begin

You must have already created the database for the UDDI registry.

About this task

Perform this task as part of setting up and deploying a new UDDI registry. The data source is used by the UDDI registry to access the UDDI database.

1. Create a J2C Authentication Data Entry. This is not required for embedded Apache Derby, but is required for network Apache Derby.
   a. Click **Security** → **Global security** → **[Authentication] Java Authentication and Authorization Service** → **J2C authentication data**.
   b. Click **New** to create a new J2C authentication data entry.
   c. Enter the following details:

      Alias A suitable short name, for example UDDIAlias.

      Userid

      The database user ID (for example db2admin for DB2, or IBMUDDI for Oracle), which is used to read and write to the UDDI registry database. For network Apache Derby, the user ID can be any value.

      Password

      The password associated with the user ID specified earlier. For network Apache Derby, the password can be any value.

      Description

      A description of the user ID.

      Click **Apply**, then save the changes to the master configuration.
2. Create a JDBC Provider, if a suitable one does not already exist, using the following table to determine the provider type and implementation type for your chosen database:

Database	Provider type	Implementation type
DB2	DB2 UDB for iSeries (Native)	Connection Pool data source
Oracle	Oracle JDBC Driver	Connection Pool data source
Embedded Apache Derby	Derby JDBC Driver	Connection Pool data source
Network Apache Derby	Derby Network Server JDBC Driver provider	Connection Pool data source
Microsoft SQL Server	DataDirect Connect JDBC Driver Microsoft SQL Server JDBC Driver	Connection Pool data source

   For details on how to create a JDBC provider, see Creating and configuring a JDBC provider using the administrative console.
3. Use the following steps to create the data source for the UDDI registry:
   a. Click **Resources** → **JDBC** → **JDBC Providers**.

b. Select the scope of the JDBC provider that you selected or created earlier, that is, the level at which the JDBC provider is defined. For example, for a JDBC provider that is defined at the level of server1, select the following:

```
Node=Node01, Server=server1
```

All the JDBC providers that are defined at the selected scope are displayed.

c. Select the JDBC provider that you created earlier.

d. Under **Additional Properties**, select **Data sources**. Do not select the **Data sources (WebSphere Application Server V4)** option.

e. Click **New** to create a new data source.

f. In the **Create a data source** wizard, enter the following data:

Name A suitable name, for example UDDI Datasource.

JNDI name
Set this value to **datasources/uddids**. This is a mandatory field.

You must not have any other data sources that use this JNDI name. If you have another data source that uses this JNDI name, you must either remove it or change its JNDI name. For example, if you created a default UDDI node previously using an Apache Derby database, before you continue, use the uddiRemove.jacl script with the default option to remove the data source and the UDDI application instance.

Component-managed authentication alias

- For DB2, Oracle, or network Apache Derby, select the alias that you created in step 2. It is prefixed by the node name, for example MyNode/UDDIAlias.
- for embedded Apache Derby leave this set to (none).

g. Click **Next**.

h. On the database-specific properties page of the wizard, enter the following data:

- for DB2:

Database name
For example:

```
*LOCAL
```

- For Oracle - **URL** - for example:

```
jdbc:oracle:oci8:@<Oracle database name>
```

- For Apache Derby (embedded or network) - **Database name** - for example:

```
profile_root/databases/com.ibm.uddi/UDDI30
```

For network Apache Derby, also make sure that the **Server name** and **Port number** match the network server.

Use this Data Source in container-managed persistence (CMP)
Ensure that the check box is cleared.

i. Click **Next**, then check the summary and click **Finish**.

j. Click the data source to display its properties, and add the following information:

Description
A description of the data source.

Category
Set this value to uddi.

Data store helper class name
This value is provided automatically:

Database	Data store helper class name
DB2	com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper
Oracle 9i or 10g	com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
Oracle 11g	com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper
Embedded Apache Derby	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
Network Apache Derby	com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper

Mapping-configuration alias
Set this option to DefaultPrincipalMapping.

k. Click **Apply**.

l. Select **Additional Properties** → **Custom Properties** → **libraries**.

m. Enter `IBMUDI30,IBMUDS30` in the **Value** field and click **OK**.

n. Save the changes to the master configuration.

4. Test the connection to your UDDI database by selecting the check box next to the data source and clicking **Test connection**. A message similar to "Test Connection for datasource UDDI Datasource on server server1 at node Node01 was successful" is displayed. If a different message is displayed, use the information in that message to investigate and resolve the problem.

What to do next

Continue with setting up and deploying your UDDI registry node.

Deploying the UDDI registry application

You deploy a UDDI registry application as part of setting up a UDDI node. You can either use the supplied script, or use the administrative console.

Before you begin

Before you deploy a UDDI registry application, you must create the database and data source for the UDDI registry.

About this task

Use this task as part of "Setting up a default UDDI node" on page 463 or "Setting up a customized UDDI node" on page 472.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the uddiDeploy.jacl script as shown, from the *app_server_root*/bin directory.

   This script deploys the UDDI registry to a server that you specify.

   ```
   wsadmin [-conntype none] [-profileName profile_name] -f uddiDeploy.jacl
                  node_name
                  server_name
   ```

   where:

   - '-conntype none' is optional, and is needed only if the application server is not running.
   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile is used.
   - *node_name* is the name of the WebSphere Application Server node on which the target server runs. The node name is case sensitive.
   - *server_name* is the name of the target server on which you wish to deploy the UDDI registry, for example, server1. The server name is case sensitive.

For example, to deploy UDDI on node 'MyNode' and server 'server1', (assuming that server1 is already started):

```
wsadmin -f uddiDeploy.jacl MyNode server1
```

You can also deploy the UDDI application (the uddi.ear file) using the administrative console, in the normal way. However, if you use the administrative console, some steps that the uddiDeploy.jacl script performs automatically do not occur. If you use the administrative console to install the UDDI application, you must perform some actions manually. To do this, use the following steps:

a. Install the application.
b. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***uddi_application*** → **[Detail Properties] Class loading and update detection**.
c. Ensure that **Class loader order** is set to **Classes loaded with application class loader first**.
d. Ensure that **WAR class loader policy** is set to **Single class loader for application**.

What to do next

Continue setting up the UDDI node.

Setting up a customized UDDI node

You set up a UDDI node with your own property values so that it is suitable for production configurations.

About this task

Use this task to set up a UDDI node with property values that you choose. You cannot change the mandatory node properties, such as node ID, after the initialization of the node. Such a node is suitable for production purposes.

1. Review the information in "Database considerations for production use of the UDDI registry" on page 461 to decide which database system to use, then create a database schema to hold the UDDI registry by completing one of the following tasks. Do not use the default node options where specified.
   - "Creating a DB2 distributed database for the UDDI registry" on page 464
   - "Creating a DB2 for iSeries database for the UDDI registry" on page 465
   - "Creating an Apache Derby database for the UDDI registry" on page 467
   - "Creating an Oracle database for the UDDI registry" on page 468
2. Set up a data source for the UDDI registry application to use to access the database, as described in "Creating a data source for the UDDI registry" on page 469.
3. Deploy the UDDI registry application, as described in "Deploying the UDDI registry application" on page 471.
4. Click **Applications** → **Application Types** → **WebSphere enterprise applications** to display the installed applications. Start the UDDI registry application by selecting the check box next to it and clicking **Start**. Alternatively, if the application server is not already running, start the application server. This action automatically starts the UDDI registry application. The UDDI node is now active.

   Note: Restarting the UDDI application, or the application server, always reactivates the UDDI node, even if the node was previously deactivated.

What to do next

Because you chose a user-customized UDDI node, you need to set the properties for the UDDI node using UDDI administration, and initialize the node, before it is ready to accept UDDI requests. See "Initializing the UDDI registry node" on page 481 for details.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

<DataBaseName>
The name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

<DB2UserID>
A DB2 userid with administrative privileges.

<DB2Password>
The password for the DB2 userid.

<BufferPoolName>
The name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, because the buffer pool is created as part of this task.

<TableSpaceName>
The name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
The name of a temporary table space. A suggested value is udditstemp, but any name can be used, because the temporary table space is created as part of this task.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt.
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`

c. `connect reset`
d. `terminate`
e. `force application all`
f. `terminate`
g. `stop`
h. `start`

7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
9. [Optional] Enter the following command if you want the database to be used as a default UDDI node:
   `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for iSeries database for the UDDI registry

Perform this task if you want to use DB2 for iSeries as the database store for your UDDI registry data.

Before you begin

IBMUDI30 and IBMUDS30 are the default names of the UDDI registry schema in the SQL scripts mentioned below. These default names are the recommended values and are assumed throughout the UDDI documentation. If you want to use different names, modify the SQL files listed below, then substitute your own names whenever you see 'IBMUDI30' or 'IBMUDS30' in other sections of the documentation.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Use iSeries Navigator to run SQL scripts.
   a. Open iSeries Navigator.
   b. Expand **My Connections** → **iSeriesName** → **Databases**.
   c. Select **iSeriesName**.

d. Right-click **Run SQL Scripts...**.

A **Run SQL Scripts** window opens.

2. Open the i5/OS DB2 SQL files.
   a. Map a network drive to the root directory of your i5/OS server's integrated file system.
   b. In Windows Explorer, expand the `WAS_HOME/UDDIReg/databaseScripts` directory.
   c. Open the following SQL files with a text editor (such as Notepad):
      - uddi30crt_10_prereq_db2_iSeries.sql
      - uddi30crt_20_tables_generic_iSeries.sql
      - uddi30crt_25_tables_db2udb_iSeries.sql
      - uddi30crt_30_constraints_generic_iSeries.sql
      - uddi30crt_35_constraints_db2udb_iSeries.sql
      - uddi30crt_40_views_generic_iSeries.sql
      - uddi30crt_45_views_db2udb_iSeries.sql
      - uddi30crt_50_triggers_db2udb_iSeries.sql
      - uddi30crt_60_insert_initial_static_data_iSeries.sql
3. Copy the text to the **Run SQL Scripts** window.
   a. In the text editor of the file `uddi30crt_10_prereq_db2_iSeries.sql`, click **Edit** → **Select All**.
   b. Click **Edit** → **Copy**.
   c. In the **Run SQL Scripts** window, click **Edit** → **Paste**.
   d. Click **Run** → **All**.
   e. After the script completes running, select all the SQL text and delete it from the **Run SQL Scripts** window.
   f. Repeat the above steps for all the SQL scripts listed in step 2.

   Note: iSeries Navigator for i5/OS V5R2 will encounter an error on the last line of the SQL script when running the `uddi30crt_25_tables_db2udb_iSeries.sql` script. This error can be ignored. i5/OS V5R2 does support running UDDI in a clustered high availability environment. For more information, see the WebSphere Application Server for i5/OS tech notes.
4. This step should only be run if you want your database to be used as a default UDDI node.
   a. Open `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 2 above.
   b. Copy and run `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 3 above.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Apache Derby database for the UDDI registry

Perform this task to use Apache Derby (embedded or network) as the database store (either local or remote) for your UDDI registry.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

arg1 The path of the SQL files. On a standard installation, this is *app_server_root*/UDDIReg/databasescripts

arg2 The path to the location where you want to install the Apache Derby database.

For example, *profile_root*/databases/com.ibm.uddi

arg3 The name of the Apache Derby database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

arg4 An optional argument, which must either be the string 'DEFAULT', or be omitted. Specify DEFAULT if you want the database to be used as a default UDDI node. This argument is case sensitive.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Apache Derby database using UDDIDerbyCreate.jar.

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/jre/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Apache Derby database already exists, you are asked if you want to recreate it. If you choose to recreate the database, your existing database is deleted and a new one is created in its place. If you choose not to recreate the database, the command exits and a new database is not created.

   Note: If the application server has already accessed the existing Apache Derby database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 978, restart the server, and run the uddiDeploy.jacl script again.
3. If you are using a remote database (which requires network Apache Derby), or you want to use network Apache Derby for other reasons, for example, if you want to use Apache Derby with a cluster, configure the Apache Derby Network Server framework. For details, see the section about managing the Derby Network Server in the Derby Server and Administration Guide.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Perform this task if you want to use Oracle as the database store for your UDDI registry data. You need only do this once for each UDDI registry, as part of setting up and deploying a UDDI registry.

About this task

The supported versions of Oracle are Version 9i[3] and Version 10g[4]

3.

Restrictions:

discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

4.

You can create a remote database using Oracle, but not a local database. Because the database is remote, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql
```

      - For Version 10g and later:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql
```

   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`
2. This last command should only be run if you want the database to be used as a default UDDI node.

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
```

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a data source for the UDDI registry

You create a data source so that the UDDI registry can use it to access the UDDI database.

Before you begin

You must have already created the database for the UDDI registry.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

About this task

Perform this task as part of setting up and deploying a new UDDI registry. The data source is used by the UDDI registry to access the UDDI database.

1. Create a J2C Authentication Data Entry. This is not required for embedded Apache Derby, but is required for network Apache Derby.
   a. Click **Security** → **Global security** → **[Authentication] Java Authentication and Authorization Service** → **J2C authentication data**.
   b. Click **New** to create a new J2C authentication data entry.
   c. Enter the following details:

      Alias A suitable short name, for example UDDIAlias.

      Userid

      The database user ID (for example db2admin for DB2, or IBMUDDI for Oracle), which is used to read and write to the UDDI registry database. For network Apache Derby, the user ID can be any value.

      Password
      The password associated with the user ID specified earlier. For network Apache Derby, the password can be any value.

      Description
      A description of the user ID.

      Click **Apply**, then save the changes to the master configuration.
2. Create a JDBC Provider, if a suitable one does not already exist, using the following table to determine the provider type and implementation type for your chosen database:

Database	Provider type	Implementation type
DB2	DB2 UDB for iSeries (Native)	Connection Pool data source
Oracle	Oracle JDBC Driver	Connection Pool data source
Embedded Apache Derby	Derby JDBC Driver	Connection Pool data source
Network Apache Derby	Derby Network Server JDBC Driver provider	Connection Pool data source
Microsoft SQL Server	DataDirect Connect JDBC Driver Microsoft SQL Server JDBC Driver	Connection Pool data source

   For details on how to create a JDBC provider, see Creating and configuring a JDBC provider using the administrative console.
3. Use the following steps to create the data source for the UDDI registry:
   a. Click **Resources** → **JDBC** → **JDBC Providers**.
   b. Select the scope of the JDBC provider that you selected or created earlier, that is, the level at which the JDBC provider is defined. For example, for a JDBC provider that is defined at the level of server1, select the following:

```
Node=Node01, Server=server1
```

      All the JDBC providers that are defined at the selected scope are displayed.
   c. Select the JDBC provider that you created earlier.
   d. Under **Additional Properties**, select **Data sources**. Do not select the **Data sources (WebSphere Application Server V4)** option.
   e. Click **New** to create a new data source.
   f. In the **Create a data source** wizard, enter the following data:

      Name A suitable name, for example UDDI Datasource.

JNDI name

Set this value to **datasources/uddids**. This is a mandatory field.

You must not have any other data sources that use this JNDI name. If you have another data source that uses this JNDI name, you must either remove it or change its JNDI name. For example, if you created a default UDDI node previously using an Apache Derby database, before you continue, use the uddiRemove.jacl script with the default option to remove the data source and the UDDI application instance.

Component-managed authentication alias

- For DB2, Oracle, or network Apache Derby, select the alias that you created in step 2. It is prefixed by the node name, for example MyNode/UDDIAlias.
- for embedded Apache Derby leave this set to (none).

g. Click **Next**.

h. On the database-specific properties page of the wizard, enter the following data:

- for DB2:

Database name

For example:

```
*LOCAL
```

- For Oracle - **URL** - for example:

```
jdbc:oracle:oci8:@<Oracle database name>
```

- For Apache Derby (embedded or network) - **Database name** - for example:

```
profile_root/databases/com.ibm.uddi/UDDI30
```

For network Apache Derby, also make sure that the **Server name** and **Port number** match the network server.

Use this Data Source in container-managed persistence (CMP)

Ensure that the check box is cleared.

i. Click **Next**, then check the summary and click **Finish**.

j. Click the data source to display its properties, and add the following information:

Description

A description of the data source.

Category

Set this value to uddi.

Data store helper class name

This value is provided automatically:

Database	Data store helper class name
DB2	com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper
Oracle 9i or 10g	com.ibm.websphere.rsadapter.Oracle10gDataStoreHelper
Oracle 11g	com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper
Embedded Apache Derby	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
Network Apache Derby	com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper

Mapping-configuration alias

Set this option to DefaultPrincipalMapping.

k. Click **Apply**.

l. Select **Additional Properties** → **Custom Properties** → **libraries**.

m. Enter `IBMUDI30,IBMUDS30` in the **Value** field and click **OK**.

n. Save the changes to the master configuration.

4. Test the connection to your UDDI database by selecting the check box next to the data source and clicking **Test connection**. A message similar to "Test Connection for datasource UDDI Datasource on server server1 at node Node01 was successful" is displayed. If a different message is displayed, use the information in that message to investigate and resolve the problem.

What to do next

Continue with setting up and deploying your UDDI registry node.

Deploying the UDDI registry application

You deploy a UDDI registry application as part of setting up a UDDI node. You can either use the supplied script, or use the administrative console.

Before you begin

Before you deploy a UDDI registry application, you must create the database and data source for the UDDI registry.

About this task

Use this task as part of "Setting up a default UDDI node" on page 463 or "Setting up a customized UDDI node" on page 472.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the uddiDeploy.jacl script as shown, from the *app_server_root*/bin directory.

   This script deploys the UDDI registry to a server that you specify.

   ```
   wsadmin [-conntype none] [-profileName profile_name] -f uddiDeploy.jacl
                  node_name
                  server_name
   ```

   where:

   - '-conntype none' is optional, and is needed only if the application server is not running.
   - '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile is used.
   - *node_name* is the name of the WebSphere Application Server node on which the target server runs. The node name is case sensitive.
   - *server_name* is the name of the target server on which you wish to deploy the UDDI registry, for example, server1. The server name is case sensitive.

   For example, to deploy UDDI on node 'MyNode' and server 'server1', (assuming that server1 is already started):

   ```
   wsadmin -f uddiDeploy.jacl MyNode server1
   ```

   You can also deploy the UDDI application (the uddi.ear file) using the administrative console, in the normal way. However, if you use the administrative console, some steps that the uddiDeploy.jacl script performs automatically do not occur. If you use the administrative console to install the UDDI application, you must perform some actions manually. To do this, use the following steps:

   a. Install the application.
   b. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***uddi_application*** → **[Detail Properties] Class loading and update detection**.
   c. Ensure that **Class loader order** is set to **Classes loaded with application class loader first**.
   d. Ensure that **WAR class loader policy** is set to **Single class loader for application**.

What to do next

Continue setting up the UDDI node.

Initializing the UDDI registry node

Use this topic to initialize a UDDI registry node after set up or migration.

Before you begin

You must have already set up a UDDI registry node, either as a new node or to use for migrating a UDDI registry Version 2 node.

About this task

The UDDI registry node has various properties, some of which must be set before initializing the node. There are two categories of UDDI registry node properties:

- Mandatory node properties. These properties must be set before the UDDI node can be initialized. You may set these properties as many times as you wish before initialization. However, once the UDDI node has been initialized, these properties will become read only for the lifetime of that UDDI node. It is very important to set these properties correctly.
- All other properties. These properties may be set before, and after, initialization.

Configure these properties and initialize the node using the UDDI administrative console or JMX management interface.

1. Click **UDDI** → **UDDI Nodes** > *UDDI_node_id* to display the properties page for the UDDI registry node.
2. Set the mandatory node properties to suitable, and valid, values. These properties are indicated by the presence of a '*' next to the input field. The properties are listed below; more information on each property is given in the context help of the administrative console.

   UDDI node ID
   This must be a text string beginning with 'uddi:' that is unique to this UDDI node. The default value may be sufficient, but if you accept it you should ensure that it is unique.

   UDDI node description
   This is a text string describing the node.

   Root key generator
   This must be a text string beginning with 'uddi:' that is unique to this UDDI node. The default value may be sufficient but may contain text, such as 'keyspace_id', that you should modify to match your system. If you accept the default value, ensure that it is unique for this UDDI node.

   Prefix for generated discoveryURLs
   This should be a valid URL.
3. If you are migrating from Version 2 of the UDDI registry, use the table below to perform the following steps:
   - Set any properties from uddi.properties that **must** remain the same as Version 2.
   - Set any properties from uddi.properties that you would like to keep the same (such as dbMaxResultCount).

Version 2 UDDI property (set in uddi.property file)	Version 3 UDDI Property (set via Administrative Console or UDDI Administrative Interface)	Recommended Version 3 UDDI property setting
dbMaxResultCount	maximum inquiry response set size	You might want to retain the value from Version 2, but can safely change this (or use the default)
persister	no equivalent	Not applicable

Version 2 UDDI property (set in uddi.property file)	Version 3 UDDI Property (set via Administrative Console or UDDI Administrative Interface)	Recommended Version 3 UDDI property setting
defaultLanguage	default language code	You are recommended to retain the value from Version 2
operatorName	UDDI node ID	You must use a valid value for the UDDI node ID. This will be applied to your Version 2 data as it is migrated.
maxSearchKeys	maximum search keys	You might want to retain the value from Version 2, but can safely change this (or use the default)
getServletURLprefix	Prefix for generated discoveryURLs	You should enter a valid value for your configuration, which should therefore be the same as the value used for Version 2.
getServletName	no equivalent	Not applicable

4. Set any other properties, such as policy values, that you wish to change from the default settings (or these can be changed at a later time). For an explanation of policies and properties see “UDDI node settings” on page 989.
5. Click **Apply** to save your changes.

   Note: You cannot change mandatory node properties after initialization. If you do not save your changes before proceeding to the initialize step, you will have to delete and recreate the database.
6. After saving your changes, initialize the UDDI node by clicking **Initialize**, at the top of the pane.

   If you are migrating from Version 2 of the UDDI registry, the Version 2 data is migrated now. The initialization may take some time to complete; to track its progress, return to the node collection page and click the refresh icon at the top of the **Status** column. Alternatively, open a second administrative console window, and use the refresh icon in the same manner. The UDDI node passes through the following states
   a. Initialization pending.
   b. Initialization in progress.
   c. Migration in progress. (This state will only occur if you are migrating.)
   d. Value set creation in progress.
   e. Activated.

What to do next

If you have migrated the node from a previous version, return to Migrating to Version 3 of the UDDI registry to verify that the migration was successful. If you have created a new node, follow the instructions in “Using the UDDI registry Installation Verification Program (IVP)” to verify that you have successfully set up the UDDI node.

Using the UDDI registry Installation Verification Program (IVP)

Use the Installation Verification Program (IVP) to verify that you have successfully deployed a UDDI registry.

Before you begin

This topic describes a simple test that you can carry out as an Installation Verification Program (IVP) to verify that you have deployed a UDDI registry successfully. You should perform this task *after* you have followed the instructions in Setting up and Deploying a new UDDI registry.

1. Open a browser window and enter the URL that accesses the UDDI registry User Interface (see Using the UDDI registry user interface).

2. Under the **Quick Find** heading on the **Find** tab, click the **Business** radio button and enter % in the **Starting with** field.
3. Click **Find**. If you have deployed your UDDI registry successfully, the detail frame displays the business entity which represents this UDDI node. You can click on the business entity to see its detail.

What to do next

As a further installation verification test, you can publish and find more UDDI entities by using the UDDI registry User Interface, or you can compile and run one or more of the UDDI registry samples available through the UDDI registry link on the Samples for WebSphere Application Server page of the IBM developerWorks® WebSphere Web site.

Changing the UDDI registry application environment after deployment

You can change the environment of the UDDI registry application after you deploy it. This means you can evaluate a UDDI registry using one database, then put it into production using a different database, or you can move from a standalone application server to a network deployment cell.

About this task

After you deploy the UDDI registry application, you might want to change its environment. For example, you might perform initial evaluation of the UDDI registry using an Apache Derby database, and then want to put the UDDI registry into production using a DB2 database, or you might want to move from a standalone application server to a network deployment cell.

1. Optional: To move from a default UDDI node to a customized UDDI node, delete the UDDI registry database and recreate it by completing one of the following tasks, ensuring that you do NOT use the default node options where specified:
   - “Creating a DB2 distributed database for the UDDI registry” on page 464
   - “Creating a DB2 for iSeries database for the UDDI registry” on page 465
   - “Creating an Apache Derby database for the UDDI registry” on page 467
   - “Creating an Oracle database for the UDDI registry” on page 468

   Note: Any data saved in the default node (policies, properties and user data) will be lost when you delete the database. If you do not want to delete the database, you can instead create an entirely new customized UDDI node in a separate application server. The default UDDI node will still exist for you to use for test purposes.
2. Optional: To change the database type for the UDDI registry, perform the following steps:
   a. Stop the UDDI registry application (click **Applications** → **Application Types** → **WebSphere enterprise applications**, select the relevant check box and click **Stop**).
   b. Either change the JNDI name of the existing datasource from datasources/uddids to another value, or delete the datasource. To display the datasource properties click **Resources** → **JDBC** → **JDBC providers >** *database_type* **JDBC Provider > [Additional Properties] Data sources >** *uddi_datasource*.
   c. Create the new database by referring to one of the following topics:
      - “Creating a DB2 distributed database for the UDDI registry” on page 464
      - “Creating a DB2 for iSeries database for the UDDI registry” on page 465
      - “Creating an Apache Derby database for the UDDI registry” on page 467
      - “Creating an Oracle database for the UDDI registry” on page 468
   d. To transfer your UDDI data, use the standard capabilities of the database products to export the data from the old database, and import it into the new one.
   e. Create the new datasource. See “Creating a data source for the UDDI registry” on page 469.
   f. Restart the UDDI registry application.

g. Check that you can access your UDDI data, then delete the old database.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

<DataBaseName>
The name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

<DB2UserID>
A DB2 userid with administrative privileges.

<DB2Password>
The password for the DB2 userid.

<BufferPoolName>
The name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, because the buffer pool is created as part of this task.

<TableSpaceName>
The name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
The name of a temporary table space. A suggested value is udditstemp, but any name can be used, because the temporary table space is created as part of this task.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt.
3. Run the following command to setup the DB2 environment variables:
   ```
   set DB2CODEPAGE=1208
   ```
4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`

b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
c. `connect reset`
d. `terminate`
e. `force application all`
f. `terminate`
g. `stop`
h. `start`

7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
9. [Optional] Enter the following command if you want the database to be used as a default UDDI node:
   `db2 -tvf uddi30crt_70_insert_default_database_indicator.sql`

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for iSeries database for the UDDI registry

Perform this task if you want to use DB2 for iSeries as the database store for your UDDI registry data.

Before you begin

IBMUDI30 and IBMUDS30 are the default names of the UDDI registry schema in the SQL scripts mentioned below. These default names are the recommended values and are assumed throughout the UDDI documentation. If you want to use different names, modify the SQL files listed below, then substitute your own names whenever you see 'IBMUDI30' or 'IBMUDS30' in other sections of the documentation.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Use iSeries Navigator to run SQL scripts.
   a. Open iSeries Navigator.
   b. Expand **My Connections** → **iSeriesName** → **Databases**.

c. Select **iSeriesName**.
d. Right-click **Run SQL Scripts...**.

A **Run SQL Scripts** window opens.

2. Open the i5/OS DB2 SQL files.
   a. Map a network drive to the root directory of your i5/OS server's integrated file system.
   b. In Windows Explorer, expand the `WAS_HOME/UDDIReg/databaseScripts` directory.
   c. Open the following SQL files with a text editor (such as Notepad):
      - uddi30crt_10_prereq_db2_iSeries.sql
      - uddi30crt_20_tables_generic_iSeries.sql
      - uddi30crt_25_tables_db2udb_iSeries.sql
      - uddi30crt_30_constraints_generic_iSeries.sql
      - uddi30crt_35_constraints_db2udb_iSeries.sql
      - uddi30crt_40_views_generic_iSeries.sql
      - uddi30crt_45_views_db2udb_iSeries.sql
      - uddi30crt_50_triggers_db2udb_iSeries.sql
      - uddi30crt_60_insert_initial_static_data_iSeries.sql
3. Copy the text to the **Run SQL Scripts** window.
   a. In the text editor of the file `uddi30crt_10_prereq_db2_iSeries.sql`, click **Edit** → **Select All**.
   b. Click **Edit** → **Copy**.
   c. In the **Run SQL Scripts** window, click **Edit** → **Paste**.
   d. Click **Run** → **All**.
   e. After the script completes running, select all the SQL text and delete it from the **Run SQL Scripts** window.
   f. Repeat the above steps for all the SQL scripts listed in step 2.

   Note: iSeries Navigator for i5/OS V5R2 will encounter an error on the last line of the SQL script when running the `uddi30crt_25_tables_db2udb_iSeries.sql` script. This error can be ignored. i5/OS V5R2 does support running UDDI in a clustered high availability environment. For more information, see the WebSphere Application Server for i5/OS tech notes.
4. This step should only be run if you want your database to be used as a default UDDI node.
   a. Open `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 2 above.
   b. Copy and run `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 3 above.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Apache Derby database for the UDDI registry

Perform this task to use Apache Derby (embedded or network) as the database store (either local or remote) for your UDDI registry.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

arg1 The path of the SQL files. On a standard installation, this is *app_server_root*/UDDIReg/databasescripts

arg2 The path to the location where you want to install the Apache Derby database.

For example, *profile_root*/databases/com.ibm.uddi

arg3 The name of the Apache Derby database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

arg4 An optional argument, which must either be the string 'DEFAULT', or be omitted. Specify DEFAULT if you want the database to be used as a default UDDI node. This argument is case sensitive.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Apache Derby database using UDDIDerbyCreate.jar.

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/jre/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Apache Derby database already exists, you are asked if you want to recreate it. If you choose to recreate the database, your existing database is deleted and a new one is created in its place. If you choose not to recreate the database, the command exits and a new database is not created.

   Note: If the application server has already accessed the existing Apache Derby database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 978, restart the server, and run the uddiDeploy.jacl script again.
3. If you are using a remote database (which requires network Apache Derby), or you want to use network Apache Derby for other reasons, for example, if you want to use Apache Derby with a cluster, configure the Apache Derby Network Server framework. For details, see the section about managing the Derby Network Server in the Derby Server and Administration Guide.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Perform this task if you want to use Oracle as the database store for your UDDI registry data. You need only do this once for each UDDI registry, as part of setting up and deploying a UDDI registry.

About this task

The supported versions of Oracle are Version 9i[5] and Version 10g[6]

5.

Restrictions:

discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

6.

You can create a remote database using Oracle, but not a local database. Because the database is remote, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql
```

      - For Version 10g and later:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql
```

   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`
2. This last command should only be run if you want the database to be used as a default UDDI node.

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
```

What to do next

Continue with setting up and deploying your UDDI registry node.

Configuring Web services applications using scripting

You can use the wsadmin scripting tool to complete the several tasks for a Web services application.

Before you begin

Develop a Web services application.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

About this task

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages.

To use the wsadmin tool to configure a Web services application, publish a Web Services Description Language (WSDL) file, or to configure policy sets:

1. Launch a scripting command.
2. Follow the steps in one of the following topics, depending on what task you want to complete:
   - Publish WSDL files.
   - Configure Web service client deployed WSDL file names.
   - Configure Web service client bindings.
   - Configure Web service client preferred port mappings .
   - Configure Web service client port information .
   - Configure the scope of a Web service port .

Results

You have configured Web services applications with the wsadmin tool.

Querying Web services with the wsadmin tool

You can use the Jython or Jacl scripting language to query for Web services properties with the wsadmin tool. Use the commands in the WebServicesAdmin group to list all Web services and attributes, find attributes of a specific Web service, determine the Web service endpoint, and determine the operation name of a Web service.

Before you begin

The wsadmin administrative scripting program supports two scripting languages, Jacl and Jython. The Jacl syntax is deprecated. This topic includes examples written only in the Jython scripting language.

Before you can complete the procedure for the commands in the WebServicesAdmin group, you must launch the wsadmin tool.

About this task

Use the following commands to query for Web services and Web service attributes. If you receive a NoItemFoundException error, the specified application, module, service, or endpoint cannot be found. Verify that all input parameters are correct.

You can optionally specify the client parameter for any command in the WebServicesAdmin command group. The client parameter indicates whether to return service providers or service clients. Specify `false` to request service providers and `true` to request service clients.

- Query the configuration for all installed Web services for all enterprise applications.

  Enter the following command. You do not need to specify the application parameter for this command.

```
AdminTask.listWebServices()
```

  This command returns all installed Web services. The command also returns the application name, module name, service name, and service type for each Web service.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [application
application1] [module webapp1.war] [type JAX-WS] ]'
```

- Query the configuration for all installed Web services for a specific enterprise application.

  Enter the following command, and specify the name of the application you want to query:

```
AdminTask.listWebServices('[–application application_name -client false]')
```

  This command returns all installed Web services for the *application_name* that you specify. The command also returns the application name, module name, service name, and service type for each Web service.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [application
application1] [module webapp1.war] [type JAX-WS] ]'
```

- Query the configuration for the Web service name and type for a enterprise application.

  Enter the following command, and specify the application name, module name, and Web service name. The **client** parameter is optional.

```
AdminTask.getWebService('[–application application_name
-module module_name -service webservice_name -client false]')
```

  The command returns the Web service name and Web service type.

  Sample output:

```
'[ [service {http://www.ibm.com}service1] [client false] [type JAX-WS] ]'
```

- Query the configuration for the Web service endpoints for a enterprise application.

  The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. Enter the following command, and specify the application name, module name, and Web service name. The **client** parameter is optional.

```
AdminTask.listWebServiceEndpoints('[–application application_name
 -module module_name -service webservice_name -client false]')
```

  This command returns the port on which the Web service is installed.

  Sample output:

```
'[logicalEndpoint QuotePort01]'
```

- Query the configuration for the Web service operation names for a enterprise application.

  Enter the following command, and specify the application name, module name, Web service name, and endpoint name. The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. The **client** parameter is optional.

```
AdminTask.listWebServiceOperations('[–application application_name -module module_name
-service webservice_name -logicalEndpoint endpoint_name -client false]')
```

  This command returns all Web service operations.

  Sample output:

```
'[operation ivt_app_op1] [operation ivt_app_op2]'
```

- Query the configuration for the service providers, endpoints, and operations from each deployed asset.

  The listServices command provides generic query functions. Use the following command to display information about service providers, endpoints, and operations for enterprise applications and Web Services Notification (WSN) clients. Each parameter is optional. If you do not specify the queryProps parameter, the command returns each service provider in your configuration. If you do not specify the expandResources parameter, the command does not return the logical endpoints or operations for each service. The following command example returns each service provider and the corresponding endpoints for the `myApplication` application:

```
AdminTask.listServices('-queryProps "[[CompositionUnit=myApplication][client=false]" -expandResources endpoint']
```

  The following command example returns each JAX-WS service provider and the corresponding endpoints and operations within a cell:

```
AdminTask.listServices('[-queryProps "[[serviceType JAX-WS][client false]]"]')
```

  This command returns the service providers that match the search query.

Sample output:

```
'[ [service {http://com/ibm/was/wssample/sei/echo/}EchoService] [assetType [J2EE Application]] [client false] [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[ [service {http://com/ibm/was/wssample/sei/echo/}EchoService12] [assetType [J2EE Application]] [client false] [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[service {http://com/ibm/was/wssample/sei/ping/}PingService] [assetType [J2EE Application]] [client false] [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS] ]

[ [service {http://com/ibm/was/wssample/sei/ping/}PingService12] [assetType [J2EE Application]] [client false] [application WSSampleServicesSei]
 [module SampleServicesSei.war] [serviceType JAX-WS]]'
```

The following command example returns each WSN service client and the corresponding endpoints and operations within a cell:

```
AdminTask.listServices('[-queryProps "[[serviceType [JAX-WS (WSN)]][clie
nt true]]" -expandResource logicalEndpoint]')
```

This command returns the service providers that match the search query.

Sample output:

```
'[ [service {http://www.ibm.com/websphere/wsn/out/remote-publisher}OutboundRemotePublisherService] [assetType [WSN Service]] [client true]
 [bus bus1] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] ]

[ [assetType [WSN Service]] [service {http://www.ibm.com/websphere/wsn/out/remote-publisher}OutboundRemotePublisherService]
 [bus bus1] [client true] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] [logicalEndpoint OutboundRemotePublisherPort] ]

[ [service {http://www.ibm.com/websphere/wsn/out/notification}OutboundNotificationService] [assetType [WSN Service]] [client true]
 [bus bus1] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] ]

[ [assetType [WSN Service]] [service {http://www.ibm.com/websphere/wsn/out/notification}OutboundNotificationService]
 [bus bus1] [client true] [WSNService wsn1] [serviceType [JAX-WS (WSN)]] [logicalEndpoint OutboundNotificationPort] ]'
```

Related tasks

Starting the wsadmin scripting client
You can use the wsadmin tool to configure and administer application servers, application deployment, and server run-time operations.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

Using scripting (wsadmin)
The WebSphere administrative (wsadmin) scripting program is a powerful, non-graphical command interpreter environment enabling you to run administrative operations in a scripting language.

WebServicesAdmin command group for the AdminTask object

Use this topic as a reference for the commands for the WebServicesAdmin group of the AdminTask object. Use these commands with your administrative scripts to list available Web services, list web services attributes, determine the endpoint configuration for a Web service, and determine a specific operation name.

Use the commands in the WebServicesAdmin group to query information for installed web services. For more information about the AdminTask object, see the Commands for the AdminTask object article.

The following commands are available for the WebServicesAdmin group of the AdminTask object.

- “getWebService” on page 492
- “listServices” on page 492
- “listWebServices” on page 494
- “listWebServicesEndpoints” on page 495
- “listWebServicesOperations” on page 496

getWebService

The getWebService command retrieves the attributes for a Web service. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the Web service. (String, required)

Optional parameters

-client
Specifies whether the Web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns Web service clients. (Boolean, optional)

Return value

Returns a list of attributes, including service name, whether the service is a provider or client, and service type. The service type attribute is only applicable for service providers.

Batch mode example usage

Using Jython string:

```
AdminTask.getWebService('[–application application_name -module module_name -service webservice_name -client false]')
```

Sample output:

```
'[[service {http://www.ibm.com}service1][type JAX-WS][client false]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.getWebService ('-interactive')
```

listServices

The listServices command queries the configuration for services, endpoints, and operations. This command provides more generic query functions than the rest of commands in this command group. It is applicable to Java™ applications as well as other assets such as Web Services Notification (WSN) clients.

Target object

None.

Optional parameters

-queryProps

Specifies properties used to locate the service provider or client of interest. For example, if you specify `[[type=JAX-WS][client=true]]`, the command returns each JAX-WS reference in the enterprise applications. (Properties, optional)

The -queryProps parameter accepts the several properties. You can use one or multiple properties to specify your query criteria. Do not mix the query properties between different asset types. For example, if you specify `application` and `bus`, the command reports an error.

- Specify the following properties with the -queryProps parameter to query enterprise applications:

Property and value	Description
`assetType=J2EE Application`	Queries each Java EE application.
`application=`*`application_name`*	Queries a specific Java EE application.
`module=`*`module_name`*	Queries a specific Java EE application module. You must specify the `application` and `module` properties to query for application modules.

- Specify the following properties with the -queryProps parameter to query for WSN clients:

Property and value	Description
`assetType=WSN Service`	Queries each WSN service client.
`bus=`*`bus_name`*	Queries a specific bus.
`WSNService=`*`WSN_service_name`*	Queries a specific WSN service. You must specify the `bus` and `WSNService` properties to query for a specific WSN service.

- Specify the following properties with the -queryProps parameter to query all assets:

Property and value	Description
`serviceType=`*`service_type`*	Queries by service type. Specify `JAX-WS` to query for Java™ API for XML-Based Web Services assets. Specify `JAX-WS (WSN)` to query for Web Services Notification assets.
`client=`*`Boolean`*	Queries for clients or providers. Specify `true` to query for clients. Specify `false` to query for providers.
`service=`*`service_name`*	Queries for the logical endpoints and operations for a specific service.

-expandResource

Specifies whether to return service names only or services and detailed resource information. Specify `logicalEndpoint` or `operation`. If you specify the `logicalEndpoint` value, the command returns the matching services and each endpoint in the services. If you specify the `operation` value, the command returns the matching services and the corresponding endpoints and operations. (String, optional)

Return value

The command returns a list of properties for each service, as well as detailed endpoint and operation information if you query for endpoints and operations.

Batch mode example usage

The following examples query each service provider in the `myApplication` application. The examples return the logical endpoints for each service because the -expandResources parameter is set as `logicalEndpoint`, as the following example output demonstrates:

```
[ [service {http://com/ibm/was/wssample/sei/echo/}EchoService] [assetType [J2EE Application]]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS] ]
[ [assetType [J2EE Application]] [service {http://com/ibm/was/wssample/sei/echo/}EchoService]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS] [logicalEndpoint EchoServicePort] ]

[ [service {http://com/ibm/was/wssample/sei/ping/}PingService] [assetType [J2EE Application]]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS] ]
[ [assetType [J2EE Application]] [service {http://com/ibm/was/wssample/sei/ping/}PingService]
 [client false] [application MyWSApplication] [module ServicesModule.war] [serviceType JAX-WS]
 [logicalEndpoint PingServicePort] ]
```

Using Jython string:

```
AdminTask.listServices('[-queryProps [[application MyWSApplication][client false]] -expandResource logicalEndpoint]')
```

Using Jython list:

```
AdminTask.listServices(['–queryProps', '[[application myApplication][client false]]', '-expandResource', 'logicalEndpoint'])
```

The following examples query each service client under the myBus bus. The examples do not return logical endpoints or operations for each service client because it does not specify the -expandResource parameter.

Using Jython string:

```
AdminTask.listServices('[–queryProps [[bus myBus][client true]] ]')
```

Using Jython list:

```
AdminTask.listServices(['–queryProps', '[[bus myBus][client true]]'])
```

Interactive mode example usage

Using Jython:

```
AdminTask.listServices('-interactive')
```

listWebServices

The listWebServices command retrieves a list of available Web services for one or all applications. If an application name is not supplied, the command lists all of the Web services. This command applies to enterprise applications only.

Target object

None.

Required parameters

None.

Optional parameters

-application
Specifies the name of the deployed enterprise application. If you do not specify this parameter, the command returns all Web services in the cell. (String, optional)

-client
Specifies whether the Web service is a provider or a client. The default value is false (service provider). When set to true, the command only returns Web service clients. (Boolean, optional)

Return value

All Web services for the application specified. For each Web service, the command returns the following attributes and corresponding values: application name, module name, service name, whether the Web service is a service provider or client, and service type. The service type is only specified if the Web service is a service provider.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServices('[–application application1 -client false]')
```

Using Jython list:

```
AdminTask.listWebServices(['–application', 'application1', '-client', 'false'])
```

Sample output:

```
'[[service {http://www.ibm.com}service1][application application1][module webapp1.war][type JAX-WS][client false]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServices('-interactive')
```

listWebServicesEndpoints

The listWebServicesEndpoints command returns a list of logical endpoints for a Web service. The logical endpoint name is the port name in the Web Services Description Language (WSDL) document. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the Web service. (String, required)

Optional parameters

-client
Specifies whether the Web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns Web service clients. (Boolean, optional)

Return value

Returns the logical endpoint name for the Web service specified.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServiceEndpoints('[–application application_name -module module_name -service webservice_name -client false]')
```

Using Jython list:

```
AdminTask.listWebServiceEndpoints(['–application', 'application_name', '-module', 'module_name', '-service', 'webservice_name', '-client', 'false'])
```

Sample output:

```
'[[logicalEndpoint QuotePort01]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServicesEndpoints('-interactive')
```

listWebServicesOperations

The listWebServicesOperations command returns a list of Web service operations. This command applies to enterprise applications only.

Target object

None.

Required parameters

-application
Specifies the name of the deployed enterprise application. (String, required)

-module
Specifies the name of the module. (String, required)

-service
Specifies the name of the Web service. (String, required)

-logicalEndpoint
The port name in the Web Services Description Language (WSDL) document. (String, required)

Optional parameters

-client
Whether the Web service is a provider or a client. The default value is `false` (service provider). When set to `true`, the command only returns Web service clients. (Boolean, optional)

Return value

Returns the operation name for the Web service specified.

Batch mode example usage

Using Jython string:

```
AdminTask.listWebServiceOperations('[–application application_name -module module_name -service webservice_name -client false
 -logicalEndpoint endpoint_name]')
```

Using Jython list:

```
AdminTask.listWebServiceOperations(['–application', 'application_name', '-module', 'module_name', '-service', 'webservice_name', '-client',
 'false', '-logicalEndpoint', 'endpoint_name'])
```

Sample output:

```
'[[operation ivt_app_op1][operation ivt_app_op2]]'
```

Interactive mode example usage

Using Jython:

```
AdminTask.listWebServicesOperations('-interactive')
```

Configuring application and system policy sets for Web services using scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

Before you begin

Develop a Web services application. For additional information, see the Web services applications topics in the information center.

If you develop an application that uses a custom policy set, the custom policy set configuration is not included in the application enterprise archive (EAR) file. Install the application and import the custom policy set separately.

About this task

The commands in the PolicySetManagement group for the AdminTask object configure both application and system policy sets. Use the following tasks to configure and manage policy sets for your Web services.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the

default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

- Use the PolicySetManagement group of commands to configure application and client policy sets:
  - Create a new policy set or copy an existing policy set.
  - Add policies to your policy set.
  - Attach your policy set to an application, Web service, endpoint, or operation.
  - Customize cell-wide, server-specific, or application binding configurations.
  - Manage and edit your policy set configurations.
    - Edit, enable, disable, or remove policies.
    - Add, edit, or remove policy set attachments.
    - Export and import policy sets.
    - Delete policy sets.
- Use the PolicySetManagement group of commands to configure system policy sets.
  - Create a new system policy set or copy an existing system policy set.
  - Add policy types for your policy set.
  - Add trust service attachments.
  - Customize binding configurations.
  - Manage and edit your policy set configurations.
    - Edit, enable, disable or remove policies.
    - Add, edit, or remove policy set attachments.
    - Export and import policy sets.
    - Delete policy sets.

Creating policy sets using the wsadmin tool

Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

Before you begin

In order to complete this task, you must use the Administrator role with cell-wide access when administrative security is enabled.

About this task

There are three ways to create a new policy set using the wsadmin tool. You can create a new policy set and its configuration, copy an existing policy set, or import a policy set.

When you create a new policy set, you must add policies. If you copy an existing policy set, you can transfer the policies and attachments that are configured on the existing policy set. The command examples in this topic use batch mode syntax. You can use the -interactive option with all commands in the PolicySetManagement group.

- Create a new policy set using the Jython scripting language.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Determine the policy requirements for your Web services.
  3. Enter the command syntax to create a new policy set with a given name.

     Based on your configuration, there are two types of policy sets to create. You can use both application and system policy sets with Java API for XML-Based Web Services (JAX-WS) applications. Use the -policySetType parameter to specify the type of policy set. To create an application policy set, specify `application` for the value of the -policySetType parameter. To create a policy set for the trust service, specify `system` or `system/trust` for the -policySetType parameter. For WS-MetadataExchange attachments, specify `system` for the -policySetType parameter. The -policySetType parameter is optional. The wsadmin tool creates an application policy set if the -policySetType parameter is not specified.

     Enter the following command to create an application policy set:

```
AdminTask.createPolicySet('[-policySet PolicySet1 -description policySet_description]')
```

     Enter the following command to create a policy set for the trust service:

```
AdminTask.createPolicySet('[-policySet PolicySet1 -description policySet_description -policySetType system]')
```

     The command returns a success or failure message.
  4. Add policies for your new policy set. Use this step to add a policy with default values for the specified policy set.

     Enter the following command to add and enable a policy:

```
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType policyType_name]')
```

     Enter the following command to add and disable a policy. Your configuration changes are contained within the policy set, but will have no effect on the system if the -enabled parameter is set to `false`.

```
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType policyType_name -enabled false]')
```

     The command returns a success or failure message. Repeat this step to create additional policies for your configuration.
  5. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

- Copy an existing policy set using the Jython scripting language.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Determine the policy requirements for your Web services.
  3. Enter the command syntax to copy an existing policy set:

     Set the -transferAttachments parameter to `true` to transfer the attachments from the existing policy set to the new policy set. The default value for the -transferAttachments parameter is `false`.

Enter the following command to create the new policy set and to transfer the attachments of the existing policy set:

```
AdminTask.copyPolicySet('[-sourcePolicySet existingPolicySet_name -newPolicySet PolicySet1 -newDescription PolicySet1_description -transferAttachments true]')
```

The command returns a success or failure message.

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

- Import a policy set from an archive file or import a default policy set using the Jython scripting language.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Determine the policy requirements for your Web services.
  3. Import a policy set.

     Use the importPolicySet command to import the archive file containing the policy set configuration of interest to the destination environment. Specify the verifyPolicySetType parameter to verify that the policy set to import matches a specific type. Set the value as `application`, `system`, or `system/trust` to specify the policy set type. You cannot import a policy set onto a server or client environment if the policy set already exists in the destination environment.

     For example, the following command creates a customSecureConversation policy set from the customSC.zip archive file:

```
AdminTask.importPolicySet('[-importFile /IBM/WebSphere/AppServer/bin/customSC.zip -verifyPolicySetType system/trust]')
```

     Additionally, you can also use the importPolicySet command to import a default policy set onto a server environment, as the following example demonstrates:

```
AdminTask.importPolicySet('[-defaultPolicySet SecureConversation -policySet copyOfdefaultSC -verifyPolicySetType system]')
```

     The command returns a success or failure message.

  4. Save the configuration changes.

     Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

If you receive a success message after entering the commands, you can now manage a policy set that is customized for your Web services applications. You can further configure the policy set and policies.

What to do next

Use the **validatePolicySet** command to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Related tasks

“Updating policy set attributes using the wsadmin tool”
Use policy sets to centrally manage policies that are customized for your Web services. Use the Jython or Jacl scripting language with the wsadmin tool to update policy set attributes. You can also query the configuration for an existing policy set and respective attributes.

“Deleting policy sets using the wsadmin tool” on page 546
Use the Jython or Jacl scripting language to delete policy sets from your configuration with the wsadmin tool. You must remove all policy set attachments before removing the policy set.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy sets using the administrative console” on page 806
You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Updating policy set attributes using the wsadmin tool

Use policy sets to centrally manage policies that are customized for your Web services. Use the Jython or Jacl scripting language with the wsadmin tool to update policy set attributes. You can also query the configuration for an existing policy set and respective attributes.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to update policy sets.
Configurator	The Configurator role cannot update policy sets.
Deployer	The Deployer role cannot update policy sets.
Operator	The Operator role cannot update policy sets.
Monitor	The Monitor role cannot update policy sets.

About this task

After creating a new policy set, you can update your policy set configuration with the updatePolicySet command. The command uses a properties object to update all attributes or a subset of attributes for the specified policy set.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. View the current attribute values for the policy set of interest.

   Enter the following command to display the attribute values of the *policySet1* policy set:

```
AdminTask.getPolicySet('[-policySet policySet1]')
```

3. Modify attributes for the policy set of interest.

   Enter the following command to modify the `type` and `description` attributes for the *policySet1* policy set.

```
AdminTask.updatePolicySet('-policySet policySet1 -attributes "[ [type application] [description [my custom policy set to manage WSSecurity]] ]"')
```

   You can also use the -interactive option to update a policy set. To start -interactive mode, enter this command:

```
AdminTask.updatePolicySet('-interactive')
```

4. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

If you received a success message after entering the commands, manage and customize the policy set for your Web services applications.

What to do next

After updating policy set attributes, you can further configure policies and policy set attachments using the commands and parameters for the PolicySetManagement group of the AdminTask object.

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Related tasks

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Deleting policy sets using the wsadmin tool” on page 546
Use the Jython or Jacl scripting language to delete policy sets from your configuration with the wsadmin tool. You must remove all policy set attachments before removing the policy set.

“Adding and removing policies using the wsadmin tool”
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy sets using the administrative console” on page 806
You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Policy configuration properties for all policies” on page 549
You can use the **attributes** parameter with the setPolicyType and setBinding commands to specify various properties for each quality of service (QoS) within a policy set. You can use the properties in this topic with each QoS within application and system policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Adding and removing policies using the wsadmin tool

You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Additionally, if administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create and remove policies.
Configurator	The Configurator role cannot create or remove policies.
Deployer	The Deployer role cannot create or remove policies.
Operator	The Operator role cannot create or remove policies.
Monitor	The Monitor role cannot create or remove policies.

About this task

Policies define which Qualities of Service (QoS) to manage within a policy set. Policy definitions are based on the standards set by the Organization for the Advancement of Structured Information (OASIS) and Web Services Security specifications.

For application policy sets, you can add the following policies:

- WSSecurity
- WSReliableMessaging
- WSAddressing
- HTTPTransport
- SSLTransport
- WSTransaction
- JMSTransport

For system policy sets, you can add the following policies:

- WSSecurity
- WSAddressing
- HTTPTransport
- SSLTransport
- WS-MetadataExchange

Use the following steps to add or remove policy types from your policy set configurations:

- Add a policy to a policy set. Use this section to add a policy with default values to the specified policy set. You can create and enable or create and disable the policy.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. List all policies for a specified policy set.

     Enter the following command and specify the policy set of interest to list all policies that have been added to the policy set:

     ```
     AdminTask.listPolicyTypes('[-policySet PolicySet1]')
     ```

     Enter the following command to list all the available policies:

     ```
     AdminTask.listPolicyTypes()
     ```
  3. Add the policy to your configuration.

     Enter the following command to add and enable a policy:

     ```
     AdminTask.addPolicyType('[-policySet PolicySet1
     -policyType policyType_name]')
     ```

     Enter the following command to add and disable a policy. Your configuration changes are contained within the policy set, these changes do not effect the system if the -enabled parameter is set to `false`.

     ```
     AdminTask.addPolicyType('[-policySet PolicySet1
     -policyType policyType_name -enabled false]')
     ```
  4. Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```
  5. For your configuration changes to take effect, restart all applications with attachments to the policy set.

  The command returns a success or failure message. Repeat this step to create additional policies for your configuration.
- Remove a policy from the policy set configuration. The deletePolicyType command removes the specified policy from the policy set. Applications with attachments to the policy set are not affected until the application restarts.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Enter the following command to list all policies for the policy set of interest:

   ```
   AdminTask.listPolicyTypes('[-policySet PolicySet1]')
   ```

3. Enter the following command to remove the policy:

   ```
   AdminTask.deletePolicyType('[-policySet PolicySet1
   -policyType policyType_name]')
   ```

   The command returns a success or failure message.
4. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

5. For your configuration changes to take effect, restart all applications with attachments to the policy set.

What to do next

Use the validatePolicySet command to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Related concepts

“Web services policies” on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

Related tasks

“Editing policy configurations using the wsadmin tool”
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to edit policy configurations for your policy sets.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Enabling policies for policy sets using the administrative console” on page 950
Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding policies to policy sets using the administrative console” on page 903
You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

“Managing policies in a policy set using the administrative console” on page 902
When working with policy sets in the administrative console, you can customize the included policies to ensure message security. You can enable, disable, customize, add, or delete policies from a policy set. With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

Related reference

“Policy configuration properties for all policies” on page 549
You can use the **attributes** parameter with the setPolicyType and setBinding commands to specify various properties for each quality of service (QoS) within a policy set. You can use the properties in this topic with each QoS within application and system policy sets.

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Editing policy configurations using the wsadmin tool

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to edit policy configurations for your policy sets.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to modify policies.
Configurator	The Configurator role cannot modify policies.
Deployer	The Deployer role cannot modify policies.
Operator	The Operator role cannot modify policies.
Monitor	The Monitor role cannot modify policies.

About this task

Policies define the type of policy to manage within a policy set. Policies are based on the Quality of Services (QoS), such as Web Services Security (WS-Security) and Web Services Addressing (WS-Addressing). Policy definitions are based on the standards set by the Organization for the Advancement of Structured Information (OASIS) and WS-Security specifications.

Use the following steps to edit existing policies in your policy set configurations:

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Determine which policy set to edit.

   To view a list of policies on a policy set, enter the listPolicyTypes command, specifying the policy set of interest.

```
AdminTask.listPolicyTypes('[-policySet PolicySet1]')
```

   Enter the listPolicyTypes command without the policySet parameter to view a list of available policies for all policy sets in your configuration:

```
AdminTask.listPolicyTypes()
```

3. Review the policy attributes to edit.

   Enter the getPolicyType command, specifying the policy and associated policy set of interest.

```
AdminTask.getPolicyType('[-policySet PolicySet1 -policyType myPolicyType]')
```

4. Modify the policy set attributes.

   Use the setPolicyType command to update the policy configuration. Update one or multiple attributes by passing a properties object for the -attributes parameter. The following example modifies the `enabled` and `provides` properties:

```
AdminTask.setPolicyType('[-policySet PolicySet1 -policyType myPolicyType
 -attributes "[[enabled true][provides security]]"')
```

5. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

6. For your configuration changes to take effect, restart all applications with attachments to the policy set.

What to do next

Use the validatePolicySet to validate your policy set configurations after modifying attributes for policies. For example, enter the following command to validate the *PolicySet1* policy set:

```
AdminTask.validatePolicySet('-policySet PolicySet1')
```

Related concepts

“Web services policies” on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

Related tasks

“Modifying policies using the administrative console” on page 905
With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Enabling policies for policy sets using the administrative console” on page 950
Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

“Adding policies to policy sets using the administrative console” on page 903
You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

“Managing policies in a policy set using the administrative console” on page 902
When working with policy sets in the administrative console, you can customize the included policies to ensure message security. You can enable, disable, customize, add, or delete policies from a policy set. With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

Related reference

“Policy configuration properties for all policies” on page 549
You can use the **attributes** parameter with the setPolicyType and setBinding commands to specify various properties for each quality of service (QoS) within a policy set. You can use the properties in this topic with each QoS within application and system policy sets.

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Enabling secure conversation using the wsadmin tool

Use this topic and the commands in the SecureConversation group of the AdminTask object to enable secure conversation client cache by creating a new policy set and bindings to attach to your applications.

Before you begin

Verify that the SecureConversation policy set is available in your configuration. By default, the SecureConversation policy set is not available. Use the importPolicySet command to import the SecureConversation policy to your configuration, as the following example demonstrates:

```
AdminTask.importPolicySet('-defaultPolicySet SecureConversation')
```

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

About this task

This topic uses the default SecureConversation policy set and default WS-Security and TrustServiceSecurityDefault bindings to enable secure conversation.

The default SecureConversation policy set contains an application policy with the symmetric binding, and a bootstrap policy with the asymmetric binding. The application policy secures application messages. The bootstrap policy secures RequestSecurityToken (RST) messages. The trust service, which issues security context token providers, uses the TrustServiceSecurityDefault system policy and the TrustServiceSecurityDefault bindings. The trust policy secures RequestSecurityTokenResponse (RSTR) messages. If you modify the bootstrap policy, you must also modify the trust policy so that both of the configurations match.

Note: Use the following steps in development and test environments only. The WS-Security bindings in this procedure contain sample key files that you must customize before using the bindings in a production environment. Create custom bindings for your production environment.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Copy the existing SecureConversation policy set.

   Use the following command example to create a new policy set by copying the existing SecureConversation policy set:

```
AdminTask.copyPolicySet('[-sourcePolicySet SecureConversation -newPolicySet CopyOfSCPolicySet]')
```

3. Change the binding for the global security domain. If you chose the **Create the server using the development template** option when you created your profile with the Profile Management Tool or the manageprofiles command utility, you can optionally skip this step.
   a. List each WS-Security policy attribute.

      To modify the binding for the global security domain, use the getDefaultBindings command to determine the binding that is set as the default for the provider or client, as the following example demonstrates:

```
AdminTask.getDefaultBinding('-bindingType provider')
```

   b. Display the attributes for the binding.

      Use the getBinding command to display the current attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-bindingLocation "" -bindingName myBinding')
```

   c. Modify the outbound configuration for the protection token.

      Use the following commands to modify the outbound configuration for the protection token:

```
cmd1_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.key.name [CN=Bob,O=IBM, C=US]] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.keystore.storepass storepass] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler
.keystore.type JCEKS] [application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler.key.alias bob]
[application.securityoutboundbindingconfig.tokengenerator_5.callbackhandler.keystore.path ${USER_INSTALL_ROOT}
/etc/ws-security/samples/enc-sender.jceks] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "" -attributes cmd1_attributes_value
-attachmentType application]')

cmd2_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore
.path ${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks] [application.securityoutboundbindingconfig
.tokengenerator_0.callbackhandler.keystore.storepass client] [application.securityoutboundbindingconfig
.tokengenerator_0.callbackhandler.key.name [CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP]] [application
.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.keypass client] [application
.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.alias soaprequester]
```

```
[application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type JKS] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "" -attributes cmd2_attributes_value
-attachmentType application]')
```

4. Optional: Modify the TrustDefaultBindings binding. If you chose the **Create the server using the development template** option when you created your profile with the Profile Management Tool or the manageprofiles command utility, you can optionally skip this step.

   If the TrustDefaultBindings are not yet customized, use the following commands to modify the TrustDefaultBindings binding:

```
cmd3_attributes_value = "[  [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore
.storepass storepass] [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.key.alias bob]
[application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore.type JCEKS] [application
.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.keystore.path ${USER_INSTALL_ROOT}/etc
/ws-security/samples/enc-sender.jceks] [application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler
.key.name [CN=Bob, O=IBM, C=US]] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "[attachmentId 2]"
-attributes cmd3_attributes_value -attachmentType system/trust]')

cmd4_attributes_value = "[ [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.path
${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-sender.ks] [application.securityoutboundbindingconfig.tokengenerator_0
.callbackhandler.keystore.storepass client] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler
.key.name [CN=SOAPRequester, OU=TRL, O=IBM, ST=Kanagawa, C=JP]] [application.securityoutboundbindingconfig.tokengenerator_0
.callbackhandler.key.keypass client] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key
.alias soaprequester] [application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type JKS] ]"

AdminTask.setBinding('[-policyType WSSecurity -bindingLocation "[attachmentId 2]"
-attributes cmd4_attributes_value -attachmentType system/trust]')
```

5. Attach the policy set and binding to the application.

   Use the attachmentType parameter for the createPolicySetAttachment command to specify if your application is a service client or a service provider. Use the following commands to attach the *CopyOfSCPolicySet* policy set to the *myTestApp* service client application:

```
AdminTask.createPolicySetAttachment('[-applicationName myTestApp -policySet CopyOfSCPolicySet
-resources WebService:/ -attachmentType client]')
```

   Use the following commands to attach the *CopyOfSCPolicySet* policy set to the *myTestApp* service provider application:

```
AdminTask.createPolicySetAttachment('[-applicationName myTestApp -policySet CopyOfSCPolicySet
-resources WebService:/ -attachmentType application]')
```

   This step automatically assigns the bindings.

Results

Your secure conversation configuration is updated in the WSSCCache.xml file located in the cell level directory.

What to do next

Manage your secure conversation configurations with the SecureConversation command group for the AdminTask object.

Related concepts

Secure conversation client cache
For both distributed and local clients, the WebSphere Application Server secure conversation client cache stores tokens on the client.

"SecureConversation default policy sets" on page 810
The SecureConversation default policy sets are based on the Web Services Secure Conversation Language (SecureConversation) standard that establishes a secure context, based on shared keys for the client and server to use for a series of messages. This standard provides a framework to define how to secure the message exchange across organizations. The SecureConversation default policy sets include the SecureConversation policy set, the Lightweight Third-Party Authentication (LTPA) SecureConversation policy set, and the Username SecureConversation policy set.

Related tasks

Configuring the Web services security distributed cache using the administrative console
You can configure the Web services security runtime to use the security distributed cache to store security tokens.

Example: Installing a Web Services Sample with the console
The product provides a Web Services sample application that you can install on a Version 7.x application server.

Managing WS-Security distributed cache configurations using the wsadmin tool

The distributed cache stores tokens on the client. Use this topic and the commands in the WSSCacheManagement group of the AdminTask object to query, update, and remove custom and non-custom properties for the distributed cache configuration.

Before you begin

Configure a policy set with WS-Security enabled.

About this task

The distributed cache stores tokens on both distributed and local clients. WebSphere Application Server supports only the security context token for the WS-Trust security token service client and the security trust service components.

You can use the administrative console or the wsadmin tool to manage your secure conversation distributed cache configuration. You can use the wsadmin tool and the Jython scripting language syntax to:

- Query your current distributed cache configuration settings.
- Set the value for the renewal time after token expiration.
- Enable or disable distributed cache for clustered servers.
- Add custom properties to your configuration.
- Remove custom properties from your configuration.
- Query the configuration for your existing distributed cache configuration.

  You can retrieve a list of your current distributed cache configuration settings and custom properties with the queryWSSDistributedCacheConfig and queryWSSDistributedCacheCustomConfig commands. There are no required or optional parameters for the query commands.

  To list all non-custom configuration settings, run the following Jython command:

```
AdminTask.queryWSSDistributedCacheConfig()
```

  To list all distributed cache custom properties, enter the following Jython command:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

- Update your secure conversation distributed cache configuration settings and custom properties.

Use the following steps to update all non-custom distributed cache configuration settings:

1. Review your existing configuration settings by running the queryWSSDistributedCacheConfig command, as the following example demonstrates:

```
AdminTask.queryWSSDistributedCacheConfig()
```

   The command returns a properties object that contains the configuration properties and values for the distributed cache configuration. The following table displays the configuration properties that the command returns:

Property	Description
tokenRecovery	Specifies whether token recovery is enabled or disabled. If the tokenRecovery property is set to `true`, the Datasource property specifies the shared data source that is assigned to the distributed cache.
distributedCache	Specifies whether distributed caching is enabled or disabled.
Datasource	Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled.
renewIntervalBeforeTimeoutMinutes	Specifies the amount of time, in minutes, that the client waits before it attempts to renew the token.
synchronousClusterUpdate	Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled.
minutesInCacheAfterTimeout	Specifies the amount of time that the token remains in the cache after the token times out.

2. Use the updateWSSDistributedCacheConfig command to enable or disable distributed cache and to modify the amount of time after token expiration when downstream calls are allowed to complete.

   The following command example enables distributed cache, and sets the `mySharedDataSource` as the shared data source for token recovery:

```
AdminTask.updateWSSDistributedCacheConfig('[-tokenRecovery true -Datasource
 mySharedDataSource -distributedCache true')
```

3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

Use the following steps to update custom properties for your distributed cache configuration:

1. Review your existing configuration settings by executing the queryWSSDistributedCacheCustomConfig command. For example:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

   The command returns a properties object that contains the name and value pairs that correspond to each custom property.

2. Use the updateWSSDistributedCacheCustomConfig command to add custom properties for your distributed cache configuration. Specify and define each custom property by passing a properties object with the -customProperties parameter using the following Jython format:

```
-customProperties [[property1 value1][property2 value2]]
```

   For example, the following command adds the `cancelActionRST` custom property and defines the value as `http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel`:

```
AdminTask.updateWSSDistributedCacheCustomConfig('[-customProperties [[cancelActionRST
 http://schemas.xmlsoap.org/ws/2005/02/trust/RST/SCT/Cancel]]]')
```

3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

- Remove custom properties from your distributed cache configuration. Use the following steps to remove custom properties from your distributed cache configuration:

  1. Review your existing configuration settings by executing the queryWSSDistributedCacheCustomConfig command. For example:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

2. Use the deleteWSSDistributedCacheConfigCustomProperties command to remove custom properties for your distributed cache configuration. Specify the custom properties to delete by passing a string array with the -propertyNames parameter. For example, the following command removes the `cancelActionRST` custom property:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('[-propertyNames
 [cancelActionRST]]')
```

3. Enter the following command to save your configuration changes:

```
AdminConfig.save()
```

Results

Your WS-Security distributed cache configuration is updated.

Related concepts

Secure conversation client cache
For both distributed and local clients, the WebSphere Application Server secure conversation client cache stores tokens on the client.

"SecureConversation default policy sets" on page 810
The SecureConversation default policy sets are based on the Web Services Secure Conversation Language (SecureConversation) standard that establishes a secure context, based on shared keys for the client and server to use for a series of messages. This standard provides a framework to define how to secure the message exchange across organizations. The SecureConversation default policy sets include the SecureConversation policy set, the Lightweight Third-Party Authentication (LTPA) SecureConversation policy set, and the Username SecureConversation policy set.

Related tasks

Configuring the Web services security distributed cache using the administrative console
You can configure the Web services security runtime to use the security distributed cache to store security tokens.

Example: Installing a Web Services Sample with the console
The product provides a Web Services sample application that you can install on a Version 7.x application server.

Configuring custom policies and bindings for security tokens using the wsadmin tool

Use the setPolicyType and setBinding commands for the AdminTask object to specify security tokens for custom policy and binding configurations.

Before you begin

Create a new custom policy set.

About this task

The following scenarios configure the custom policy and bindings to use a Kerberos token based on the Oasis Kerberos Token Profile V1.1 specification. You can also use the setPolicyType and setBinding commands to configure other binary security tokens, such as username tokens, Lightweight Third-Party Authentication (LTPA) and SecureConversation.

- Configure custom policies for security tokens.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Display the properties of the policy of interest.

     Use the getPolicyType command to display detailed property information for the WS-Security policy type, as the following command demonstrates:

```
AdminTask.getPolicyType('-policySet AuthenticationTokenService -policyType
 WSSecurity')
```

The getPolicyType command returns a properties object that contains name and value pairs for each property, as the following sample output displays:

```
'[ [SupportingTokens.request:krb_token.CustomToken.IncludeToken
 http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient] [enabled true] [type WSSecurity]
 [description [Policies for sending security tokens and providing message confidentiality and integrity, based on the OASIS Web
 Service Security and Token Profiles specifications.]] [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri ]
 [provides ] [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ] ]'
```

3. Specify the authentication token for the policy type.

   Use the setPolicyType command to specify the Uniform Resource Identifier (URI) of the authentication token for services as the value for the `SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri` property. Use the `[ ]` syntax to specify an empty string. The following example specifies an empty string as the value for the authentication token:

```
AdminTask.setPolicyType('-policySet AuthenticationTokenService -policyType
 WSSecurity -attributes "[ [SupportingTokens.request:krb_token.CustomToken.IncludeToken
 http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient] [enabled true] [type
 WSSecurity] [description  [Policies for sending security tokens and providing message confidentiality and integrity,
 based on the OASIS Web Service Security and Token Profiles specifications.]]
 [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.uri []] [provides []]
 [SupportingTokens.request:krb_token.CustomToken.WssCustomToken.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ] ]"')
```

- Configure custom bindings for security tokens.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Display the properties of the bindings of interest.

     Use the getBinding command to display detailed property information for the binding of interest, as the following command demonstrates:

```
AdminTask.getBinding('-policyType WSSecurity -bindingLocation "" -bindingName
 AuthenticationTokenService')
```

     The getBinding command returns a properties object that contains name and value pairs for each property, as the following sample output displays:

```
'[ [application.securityinboundbindingconfig.tokenconsumer_0.properties_0.name
 com.ibm.wsspi.wssecurity.krbtoken.serviceSPN]  [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri ]
 [application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
 com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler]  [application.name
 application][application.securityinboundbindingconfig.tokenconsumer_0.properties_0.value HTTP/derekho1.firehorse.austin.ibm.com]
 [application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname system.wss.consume.KRB5BST]
 [application.securityinboundbindingconfig.tokenconsumer_0.name
 con_krbtoken][application.securityinboundbindingconfig.tokenconsumer_0.classname
 com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer]
 [application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference request:krb_token] ]'
```

  3. Specify the authentication token for the policy type.

     Use the setBinding command to specify the Uniform Resource Identifier (URI) of the authentication token for services as the value for the `application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri` property. Use the `[ ]` syntax to specify an empty string. The following example specifies an empty string as the value for the authentication token:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation ""
 -bindingName AuthenticationTokenService -attributes  "[
 [application.securityinboundbindingconfig.tokenconsumer_0.properties_0.name com.ibm.wsspi.wssecurity.krbtoken.serviceSPN]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
 http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ]
 [application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri []]
 [application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
 com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler] [application.name
 application][application.securityinboundbindingconfig.tokenconsumer_0.properties_0.value
 HTTP/derekho1.firehorse.austin.ibm.com] [application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname
 system.wss.consume.KRB5BST][application.securityinboundbindingconfig.tokenconsumer_0.name
 con_krbtoken][application.securityinboundbindingconfig.tokenconsumer_0.classname
 com.ibm.ws.wssecurity.wssapi.token.impl.CommonTokenConsumer]
 [application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference request:krb_token]
 ]"')
```

Results

If the setPolicyType and setBinding commands return a 'true' value, the system successfully updated the policy and binding configurations.

Related concepts

"WSSecurity default policy sets" on page 813
The WSSecurity default policy sets are based on the Web Services Security (WS-Security) 1.0 and Web Services Addressing (WS-Addressing) specifications. The WSSecurity default policy sets include the WSSecurity default policy set, the Lightweight Third-Party Authentication (LTPA) WSSecurity policy set, and the Username WSSecurity policy set. Use the WSSecurity default policy sets to build secure Web services.

Related tasks

"Creating policy sets using the wsadmin tool" on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

"Adding and removing policies using the wsadmin tool" on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

Related reference

"PolicySetManagement command group for the AdminTask object" on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

"WSSecurity policy and binding properties" on page 549
Use the **attributes** parameter for the setPolicyType and setBinding commands to specify additional configuration information for the WSSecurity policy and binding configurations. Application and system policy sets can use the WSSecurity policy and binding configuration.

Creating policy set attachments using the wsadmin tool

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can create policy set attachments for application resources only.
Operator	The Operator role cannot create policy set attachments.
Monitor	The Monitor role cannot create policy set attachments.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

About this task

To use a new policy set to manage policies for your application, you must attach the policy set to an application artifact or artifacts. When the application restarts, the application uses the policies from the newly attached policy set.

1. Launch a scripting command.
2. Select an application with Web services to update. Use the listWebServices command to list all Web services and the associated applications. Enter the following command to list all Web services and attributes:

```
AdminTask.listWebServices()
```

   For each Web service, the command returns the associated application name, module name, service name, and service type. For example, the following information is returned:

```
'[ [service {http://www.ibm.com}service1] [client false] [application  application1]
 [module webapp1.war] [type JAX-WS] ]'
```

3. Create a policy set attachment for an application.

   For the commands in the PolicySetManagement group, the term **resource** refers to a Web service artifact. For application and service client policy sets, the artifacts use the application hierarchy. The application hierarchy includes a Web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter.

   Note: When attempting to connect to a Web service from a thin client, verify that the resources you are specifying are valid before running the updatePolicySetAttachment command. No configuration changes are made if the requested resource does not match a resource in the attachment file for the application.

   Use the following format for application and client policy set attachments:

   - `WebService:/`

     Attaches all artifacts in the application to the policy set.
   - `WebService:/webapp1.war:{http://www.ibm.com}myService`

     Attaches all artifacts within the Web service `{http://www.ibm.com}myService` to the policy set. You must provide a fully qualified name (QName) for the service.
   - `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA`

     Attaches all operations for the `endpointA` endpoint to the policy set.
   - `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1`

     Attaches only the `operation1` operation to the policy set.

   The format for the -resource string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

   - `Trust.opName:/`

     The `opName` attribute can be issue, renew, cancel, or validate.
   - `Trust.opName:/url`

     The `opName` attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the `url` attribute.

   a. Enter the command to attach the policy set to the application. This command attaches the `policyset1` application policy set to all artifacts in the `WebService` application.

      Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy

set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

To attach a policy set to a Web service application, specify the `provider` value for the -attachmentType parameter:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -applicationName WebService -attachmentType provider]')
```

To attach a policy set to a service client application, specify the `client` value for the -attachmentType parameter, as the following example demonstrates:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -applicationName WebService -attachmentType client]')
```

To create a trust service attachment for a system policy set, specify the `provider` value for the -attachmentType parameter and the `[systemType trustService]` value for the -attachmentProperties parameter, as the following example demonstrates:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources
 "WebService:/" -applicationName WebService -attachmentType client -attachmentProperties "[systemType
 trustService]"]')
```

This command returns an attachment ID number that you must use to reference this attachment. In the next step, use the attachment ID number to set the binding configuration. For this example, the attachment ID number is `124`.

4. Run the command to set the binding. The following example demonstrates how to set the timestamp expiration attribute on the `SecureConversation123binding` binding for the `WSSecurity` policy, on the `WebService` Web service application. To attach a policy set to a Web services application, specify the `provider` value for the -attachmentType parameter.

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[ [application WebService] [attachmentId 124] ]" -attachmentType provider
 -bindingName SecureConversation123binding -attributes "[application.securityoutboundbindingconfig.timestampexpires.expires 5]"')
```

5. Save the configuration changes.

   Enter the following command to save your changes:

```
AdminConfig.save()
```

Results

Your have attached the policy set to the application artifact or artifacts specified. Restart your application to use the policies from the newly attached policy set.

What to do next

Manage and update your attachments.

Related tasks

Configuring attachments for the trust service using the administrative console
You can attach the trust service operations for a service endpoint to a system policy set and binding. Each new endpoint that is specified initially has the following four operations: issue, renew, cancel, and validate. By default, all endpoints inherit the policy set and binding that are attached to the respective trust service operation under Trust Service Defaults. However, you can explicitly attach a different policy set.

"Managing policy set attachments using the wsadmin tool"
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

"Configuring application and system policy sets for Web services using scripting" on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

"Creating policy sets using the wsadmin tool" on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

"Adding and removing policies using the wsadmin tool" on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

"Removing policy set attachments using the wsadmin tool" on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

"Search attached applications collection" on page 957
Use this page to search for applications and other resources that are attached to a specific policy set or to search for applications and other resources that have attached service resources.

"PolicySetManagement command group for the AdminTask object" on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

"WebServicesAdmin command group for the AdminTask object" on page 491
Use this topic as a reference for the commands for the WebServicesAdmin group of the AdminTask object. Use these commands with your administrative scripts to list available Web services, list web services attributes, determine the endpoint configuration for a Web service, and determine a specific operation name.

Managing policy set attachments using the wsadmin tool

Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to manage policy set attachments. If you have access to a specific resource only, you can manage policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can manage policy set attachments for application resources only.
Operator	The Operator role cannot manage policy set attachments.
Monitor	The Monitor role cannot manage policy set attachments.

About this task

Policy set attachments define how a policy set is attached to resources and binding configurations.

- Query the configuration for policy set attachments and attachment properties.

  Before making configuration changes to your policy set attachments, use the listAttachmentsForPolicySet and getPolicySetAttachments commands to view current configuration information about your policy set attachments.

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Use the listAttachmentsForPolicySet command to view all applications to which a specific policy set is attached, for example:

     ```
     AdminTask.listAttachmentsForPolicySet('[-policySet PolicySet1]')
     ```

     Use the -attachmentType parameter to narrow your query. You can query for `provider` or `client` attachments.

     Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.
  3. Use the getPolicySetAttachments command to view the properties for all policy set attachments in a specified application, for example:

     ```
     AdminTask.getPolicySetAttachments('[-applicationName application1]')
     ```

     Use the -attachmentType parameter to narrow your query. You can query for `provider` or `client` attachments.
- Determine the assets to which a specific policy set is attached.

  Use the listAssetsAttachedToPolicySet command to display the assets that are attached to the policy set of interest, as the following example demonstrates:

  ```
  AdminTask.listAssetsAttachedToPolicySet('[-policySet SecureConversation]')
  ```

  The command returns a list of properties that describe each asset. Each properties object contains the assetType property, which specifies the type of asset.
- Modify resources that apply to a policy set attachment.
  1. Launch the wsadmin scripting tool using the Jython scripting language.

2. Determine the resource of interest and review the command syntax for the updatePolicySetAttachment command.

   For the commands in the PolicySetManagement group, the term **resource** refers to a Web service artifact. For application and service client policy sets, the artifacts use the application hierarchy. The application hierarchy includes a Web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter.

   Note: When attempting to connect to a Web service from a thin client, verify that the resources you are specifying are valid before running the updatePolicySetAttachment command. No configuration changes are made if the requested resource does not match a resource in the attachment file for the application.

   Use the following format for application and client policy set attachments:

   – `WebService:/`

     Attaches all artifacts in the application to the policy set.

   – `WebService:/webapp1.war:{http://www.ibm.com}myService`

     Attaches all artifacts within the Web service `{http://www.ibm.com}myService` to the policy set. You must provide a fully qualified name (QName) for the service.

   – `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA`

     Attaches all operations for the `endpointA` endpoint to the policy set.

   – `WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1`

     Attaches only the `operation1` operation to the policy set.

   The format for the -resource string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

   – `Trust.opName:/`

     The `opName` attribute can be issue, renew, cancel, or validate.

   – `Trust.opName:/url`

     The `opName` attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the `url` attribute.

3. Modify the attachment.

   For example, the policy set attachment is connected to the operation1 operation, which is a specific single operation. To attach the 124 attachment to all operations for the endpointA endpoint, enter the following command:

   ```
   AdminTask.updatePolicySetAttachment('[-attachmentId 124 -resources
   "WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA" -applicationName application1]')
   ```

   Note: The updatePolicySetAttachment command replaces all existing resources for an attachment with the resources specified in the command. You can also update your policy set attachments using the addToPolicySetAttachment command to add resources to an existing attachment or the createPolicySetAttachment command to create an attachment for a specific resource. For more information about these commands reference the commands for the PolicySetManagement group for the AdminTask object.

4. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

- Remove resources that apply to a policy set attachment.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Determine which resources to remove with the command. You can remove a resource for each Web service artifact, each operation for an endpoint, or for a specific operation. In the following example, the command removes the `newAttach` attachment from `operation1`, which is associated with the `plantShop` application.

```
AdminTask.removeFromPolicySetAttachment('[-attachmentId newAttach -resources
"WebService:/webapp1.war:{http://www.ibm.com}myPlantService/endpointA/operation1" -applicationName plantShop]')
```

The command returns a success or failure message.

3. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

- Transfer attachments from one policy set to another policy set. This command detaches each Web service from the source policy set and attaches those Web services to the destination policy set. The destination policy set must have the same set of enabled policy types as the source policy set.

  1. Enter the following command to transfer all attachments:

     ```
     AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet PolicySet1
     -destinationPolicySet PolicySet2]')
     ```

     The command returns a success or failure message.

  2. Save the configuration changes.

     Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```

Related tasks

Configuring attachments for the trust service using the administrative console
You can attach the trust service operations for a service endpoint to a system policy set and binding. Each new endpoint that is specified initially has the following four operations: issue, renew, cancel, and validate. By default, all endpoints inherit the policy set and binding that are attached to the respective trust service operation under Trust Service Defaults. However, you can explicitly attach a different policy set.

"Configuring application and system policy sets for Web services using scripting" on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

"Creating policy set attachments using the wsadmin tool" on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

"Managing policy set attachments using the wsadmin tool" on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

"Removing policy set attachments using the wsadmin tool" on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

"Search attached applications collection" on page 957
Use this page to search for applications and other resources that are attached to a specific policy set or to search for applications and other resources that have attached service resources.

"PolicySetManagement command group for the AdminTask object" on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Configuring general, cell-wide bindings for policies using the wsadmin tool

You can use the Jython or Jacl scripting language to customize your cell-wide default binding configuration. Create multiple cell-wide general bindings that you can attach to applications.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

.

About this task

Bindings are environment and platform-specific information such as key store information, keys used for signature and encryption, or authentication information.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that

are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Determine the policy to update.

   To view a list of all available policies for a specific policy set, use the listPolicyType command. For example:

```
AdminTask.listPolicyTypes('[-policySet PolicySet1]')
```

3. Retrieve the current binding configuration for the policy to determine the attributes to update.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific policy binding. Specify a Properties object for the -bindingLocation parameter using an empty Properties object. For example:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1')
```

   To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if the WSAddressing policy has workload management enabled:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1 -attributes "[preventWLM]"')
```

   The command returns a properties object which contains the value of the requested attribute, preventWLM.

4. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a cell-wide binding, set the -bindingLocation parameter by passing a null or empty Properties object and specify the name of the binding with the -bindingName parameter. You can further customize your binding with the following parameters:

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update.	Properties, optional
-replace	Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is false.	Boolean, optional

Parameter	Description	Data type
-remove	Use this parameter to remove a specific policy from the binding configuration. The default value for the remove parameter is false. If the policyType parameter is not specified, the command removes the custom binding from the attachment. To delete the binding configuration, provide a value for the bindingName parameter and an asterisk character (*) for the attachmentId.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

You must use the -attributes parameter when editing your binding configuration for cell-wide bindings. The following example disables workload management within the cell-wide default binding for the `WSAddressing` policy:

```
AdminTask.setBinding('-policyType WSAddressing -bindingLocation "" -bindingName cellWideBinding1 -attributes "[preventWLM false]"')
```

5. Save your configuration changes.

```
AdminConfig.save()
```

Related tasks

“Reassigning bindings to policy sets” on page 856
After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“Policy set bindings settings” on page 866
Use this page to view or define general or application specific bindings configuration information that is specific to a system for policies that you can associate with the selected policy set. Use the links on this page to work with bindings for each specific policy.

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Configuring Version 6.1 server-specific default bindings for policies using the wsadmin tool

You can use the Jython or Jacl scripting language to customize WebSphere Application Server Version 6.1 server-specific default bindings for policies to match your installation environment or requirements.

Before you begin

Server-specific default bindings use the WebSphere Application Server Version 6.1 namespace.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.

Administrative role	Authorization
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Determine the policy to update.

   To view a list of all available policies for a specific policy set, use the listPolicyTypes command, as the following example demonstrates:

```
AdminTask.listPolicyTypes('[-policySet WSAddressing]')
```

3. Retrieve the current binding configuration for the policy to determine the attributes to update.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific policy binding. Specify a Properties object for the -bindingLocation parameter using the property names `node` and `server`. For example:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[node node1]
[server server1]]"')
```

   To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if the policy is enabled:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[node node1]
[server server1]]" -attributes "[[preventWLM]]"')
```

   The command returns a properties object which contains the value of the requested attribute, `preventWLM`.

4. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a server-specific default binding, set the -bindingLocation parameter using the `node` and `server` property names in a Properties object. You can further customize your binding with the following optional parameters:

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-remove	Use this parameter to remove a server-level binding configuration. The default value for the -remove parameter is `false`.	Boolean, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update. The -attributes parameter is not required if you are removing your server-level binding.	Properties, optional
-replace	Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is `false`.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

   You should always specify the -attributes parameter when editing your configuration. The following example disables workload management within the server-specific default binding for the `WSAddressing` policy:

```
AdminTask.setBinding('-policyType WSAddressing -bindingLocation "[ [server server1] [node node01] ]" -attributes "[preventWLM false]"')
```

5. Save your configuration changes.

```
AdminConfig.save()
```

Related tasks

“Creating application specific bindings for policy set attachment” on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Configuring application-specific and system bindings using the wsadmin tool

Use the Jython or Jacl scripting language to edit custom application bindings and system bindings for policies to match your installation environment or system requirements.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can edit binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Binding configurations are environment- and platform-specific information such as keystore information, keys used for signature and encryption, or authentication information. You can use the default binding for each policy set or define application-specific bindings within an application.

There are three types of bindings to use with your policy sets, including cell-level, application server level, and application-level. Default bindings are used at the cell-level or application server level. This topic refers to system binding information or bindings that are defined at the application level, which overrides the cell-level or application server level definition.

Use default bindings only to develop and test applications. You must change signing and encryption keys before using your bindings in a production environment.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.

- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Retrieve the current binding data for the attachment of interest.

   Use the getPolicySetAttachments command to determine the attachment ID. You will need to specify the attachment ID in the getBinding and setBinding commands to specify that this is a application-specific binding configuration. Use the following command to retrieve the attachment ID:

   ```
   AdminTask.getPolicySetAttachments('-applicationName application1')
   ```

   Use the getBinding command to display a properties object that contains each configuration attribute for a specific policy binding configuration. For application and client policy set attachments, specify a properties object for the -bindingLocation parameter using the `application` and `attachmentId` property names. For a system policy set attachment for the trust service, specify only the `attachmentId` property name. The following example queries for an application policy set binding configuration:

   ```
   AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 123]]"')
   ```

   To return a specific configuration attribute for the policy, use the -attributes parameter.
3. Edit the binding configuration.

   Use the setBinding command to update your binding configuration for a policy. To specify that you are editing a application-specific binding configuration, set the -bindingLocation parameter by specifying the `application` and `attachmentId` property names in a properties object. You can additionally specify the -attachmentType parameter as `provider` or `client`.

   Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

Customize your binding configuration with the following optional parameters:

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-remove	Use this parameter to remove a specific policy from the binding configuration. The default value for the -remove parameter is false. If the -policyType parameter is not specified, the command removes the application-specific binding from the attachment. To delete the binding configuration, provide a value for the -bindingName parameter and an asterisk character (*) for the -attachmentId parameter.	Boolean, optional
-attributes	Specifies the attribute values to update. This parameter can include each binding configuration attribute for the policy or a subset of attributes to update. If you do not specify the attributes parameter, the command only updates the binding configuration location that the specified attachment uses.	Properties, optional
-bindingName	Specifies the name for the binding configuration. Use this parameter to specify a name for the binding when you create a new application-specific binding. You can also use this parameter to switch an attachment to use a different, existing application-specific binding configuration. Lastly, you must specify a value for this parameter to delete a binding configuration.	String, optional
-replace	Specifies whether to replace all of the existing binding configuration attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is `false`.	Boolean, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

The following example disables workload management for the `myApplication` application's binding configuration for the `WSAddressing` policy:

```
AdminTask.setBinding('[-policyType WSAddressing -bindingLocation "[ [application myApplication] [attachmentId 123] ]"
 -attributes "[preventWLM false]" -attachmentType provider]')
```

4. Save the configuration changes.

   Enter the following command to save your changes.

   ```
   AdminConfig.save()
   ```

Related tasks

“Creating application specific bindings for policy set attachment” on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Creating application-specific and trust service-specific bindings using the wsadmin tool

You can use the Jython or Jacl scripting language to create application-specific and trust service-specific bindings to match your installation environment or requirements.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure bindings for the resource for which you have access. Only the Administrator role can configure binding attributes.
Configurator	The Configurator role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings, but cannot edit attributes.
Operator	The Operator role can view, but cannot configure bindings.
Monitor	The Monitor role can view, but cannot configure bindings.

About this task

Policy set bindings specify the details about how your quality of service (QoS) is configured. For example, a policy set attachment determines that sign, encrypt, or reliable messaging should be enabled. The policy set binding specifies how the protection is configured, for example, the path of the keystore file, the class name of the token generator, or the Java™ Authentication and Authorization Service (JAAS) configuration name.

For application policy sets, you can specify the policy set bindings at the cell-level using default binding configurations, at the application level using application-specific binding configurations, or at the cell-level with general bindings. Server-level default bindings are deprecated. If no binding information is specified during policy set attachment, the policy set inherits the default binding. In WebSphere Application Server Version 7.0, you can specify a general binding as the default for a server instead of server-default bindings.

For system policy sets, you can specify the bindings at the cell-level and the server-level. The available bindings for system policy sets are the TrustServiceSymmetricDefault and TrustServiceSecurityDefault bindings. If no custom binding information is specified by the attachment, the resources inherit the TrustServiceSymmetricDefault or TrustServiceSecurityDefault binding.

Note: Only use default binding for development and testing. You must customize the signing and encryption keys in your binding configurations for a production environment.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Use the following guidelines to manage bindings in your environment:

- To display or modify default Version 6.1 bindings, Version 7.0 trust service bindings, or to reference bindings by attachment for an application, specify the attachmentId and bindingLocation parameters with the getBinding or setBinding commands.
- To use or modify general Version 7.0 bindings, specify the bindingName parameter with the getBinding or setBinding commands.
- To display the version of a specific binding, specify the **version** attribute for the getBinding command.

Use a Version 6.1 binding for an application in a Version 7.0 environment if:

- The module in the application is installed on at least one Web Services Feature Pack server.
- The application contains at least one Version 6.1 application-specific binding. The application server does not assign general bindings to resource attachments for applications that are installed on a Web Services Feature Pack server. All application-specific bindings for an application must be at the same level.

General service provider and client bindings are not linked to a particular policy set and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers. You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The order of precedence from lowest to highest that the application server uses to determine which default bindings to use is as follows:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided Provider sample as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided Client sample as the default service client binding, you can select an existing general client binding or create a new general client binding.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Determine the type of binding to create.

   You can create application policy set bindings at the cell-level, server-level, or application-level, and trust service policy set bindings at the cell-level or server-level.
3. Retrieve the current binding configuration for the policy of interest.

   Use the getBinding command to display a Properties object containing all configuration attributes for a specific binding. Specify the location of the binding by passing a properties object using the -bindingLocation parameter and the following reference table:

Type of binding	**-bindingLocation parameter value**
Cell-level	`-bindingLocation ""`
Server-level (deprecated)	`-bindingLocation "[[node node1][server server1]]"`
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service	`-bindingLocation "[[systemType trustService] [attachmentId 123]]"`

Type of binding	-bindingLocation parameter value
WS-Notification client	-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"

For this example, the command displays the current binding configuration for the WSAddressing policy, with the 123 attachment ID, for the application1 application:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 123]]"')
```

To return a specific configuration attribute for the policy, use the -attributes parameter. For example, enter this command to determine if workload management is enabled:

```
AdminTask.getBinding('-policyType WSaddressing -bindingLocation "[[application application1][attachmentId 123]]" -attributes "[preventWLM]"')
```

The command returns a properties object which contains the value of the requested attribute, preventWLM. You might receive an error message if the binding does not exist in your configuration.

4. Create a new application-specific binding for the policy of interest.

   Use the setBinding command to create a binding configuration for a policy. To specify that you are creating an application-specific binding, set the -bindingLocation parameter by passing the application and attachmentId property names in a properties object. If you are creating a system policy set binding for the trust service, you only need to specify the attachmentId property name. You can further customize your binding with the following parameters:

Parameter	Description	Data type
-policyType	Specifies the policy of interest.	String, optional
-attachmentType	Specifies the type of policy set attachment. If the attachment is for an application, you do not need to specify this parameter. **Note:** The application and system/trust values for the -attachmentType parameter are deprecated. Specify the provider value in place of the application value. For system policy set attachments, specify the provider value for the attachmentType parameter and the "[systemType trustService]" value for the -attachmentProperties parameter. For WSNClient attachments, specify the client value for the attachmentType parameter and the bus and WSNService properties with the -attachmentProperties parameter.	String, optional
-attributes	Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset of attributes.	Properties, optional
-bindingName	Specifies the name for your new application-specific binding. A name is generated if it is not specified.	String, optional
-domainName	Specifies the domain name for the binding. Use this parameter to scope a binding to a domain other than the global security domain.	String, optional

The following example creates the WSAddressing1234binding attachment-specific binding for the WSAddressing policy, assigned to the application1 application attachment 123, and enables workload management:

```
AdminTask.setBinding('-policyType WSAddressing -bindingName
 WSAddressing123binding -bindingLocation "[ [application application1] [attachmentId 123] ]" -attributes
 "[preventWLM false]"')
```

5. Optional: Add application-specific binding properties.

   Use the setBinding command to add any additional custom properties for your application-specific binding. The application server provides custom properties that are specific to each quality of service. Use the following format to specify custom properties for the binding:

```
AdminTask.setBinding('[-bindingLocation "[ [application application1] [attachmentId 123]
 ]" -policyType WSAddressing -attributes "[[properties_x:name key_value] [properties_x:value
 value]"]')
```

6. Save your configuration changes.

```
AdminConfig.save()
```

Related tasks

“Creating application specific bindings for policy set attachment” on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Deleting application-specific bindings from your configuration using the wsadmin tool

You can use the Jython or Jacl scripting language to delete a custom application or system policy set binding from your configuration. You cannot delete cell-level default bindings.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to modify bindings. If you have access to a specific resource only, you can modify bindings for the resource for which you have access.
Configurator	The Configurator role cannot modify bindings.
Deployer	The Deployer role cannot modify bindings.
Operator	The Operator role cannot modify bindings.
Monitor	The Monitor role cannot modify bindings.

About this task

Policy set bindings specify the details about how your quality of service (QoS) is configured. For example, a policy set attachment determines that sign, encrypt, or reliable messaging is enabled. The policy set binding specifies how the protection is configured, for example, the path of the keystore file, the class name of the token generator, or the Java Authentication and Authorization Service (JAAS) configuration name.

For application policy sets, policy set bindings exist at the cell-level and server-level using default binding configurations, or at the application level using application-specific binding configurations. You can also specify cell-level general bindings. For system policy sets, bindings exist at the cell level and server level, or you can create application-specific bindings.

Use the following procedure to delete application-specific bindings for trust policy sets and application level bindings for application policy sets:

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Retrieve the current binding configuration for the policy of interest.

   Use the getBinding command to display a properties object that contains all configuration attributes for a specific binding. Specify the location of the binding by passing a properties object using the bindingLocation parameter and the following reference table:

Type of Binding	Value for the -bindingLocation parameter
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service	`-bindingLocation "[[attachmentId 123]]"`
WS-Notification client	`-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"`
General binding	`-bindingLocation []`

   In this example, the command displays the current binding configuration for the `WSAddressing` policy, with the `123` attachmentId, for the `application1` application:

```
AdminTask.getBinding('[-policyType WSAddressing -bindingLocation  "[[application
 application1][attachmentId 123]]"]')
```

   To display general policy set bindings, identify the bindings by specifying the -bindingName parameter, as the following example demonstrates:

```
AdminTask.getBinding('[-bindingLocation [] -attachmentType application
 -bindingName "General Provider Binding"]')
```

3. Remove the binding of interest from each attachment.

   You cannot remove a binding from your configuration if that binding is referenced by one or more attachments. Modify and use the following example command to remove a binding from an attachment:

```
AdminTask.setBinding('[-bindingLocation "[[application application1][attachmentId
 123]]"  -remove true]')
```

4. Delete the binding of interest.

   Use the setBinding command to delete a application-specific binding configuration. Specify the binding of interest with the -bindingName parameter, an asterisk (*) for the -attachmentId property, and set the -remove parameter to `true`. The following example setBinding command removes the `WSAddressing123binding` application policy set binding:

```
AdminTask.setBinding('[-attachmentType application -bindingName
 WSAddressing123binding  -bindingLocation "[[application application1][attachmentId *]]" -remove
 true]')
```

   The following example setBinding command removes the `customTrust` trust service binding:

```
AdminTask.setBinding('[-attachmentType "system/trust" -bindingName
 customTrust  -bindingLocation "[attachmentId *]" -remove true]')
```

   The following example setBinding command removes the `General Provider Binding` general binding:

```
AdminTask.setBinding('[-attachmentType application -bindingName "General Provider
 Binding" -bindingLocation [] -bindingScope domain -remove true]')
```

Note: You cannot delete general bindings if an attachment references the binding, or if the binding is set as the default for a server or domain.

5. Save your configuration changes.

Results

The application-specific binding of interest is removed from your configuration.

Related tasks

“Removing policy set bindings using the wsadmin tool” on page 540
You can use the Jython or Jacl scripting language to remove binding configurations for policies and resources to match your installation environment or requirements.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Importing and exporting policy sets to client or server environments using scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to import and export policy sets.
Configurator	The Configurator role cannot import and export policy sets.
Deployer	The Deployer role cannot import and export policy sets.
Operator	The Operator role cannot import and export policy sets.
Monitor	The Monitor role cannot import and export policy sets.

About this task

You can use the exportPolicySet and importPolicySet commands to exchange system or application policy sets between servers or between a client and a provider. To reuse a policy set on a new server or client, export the policy set to an archive file, then import the archive file on the destination server or client. This topic provides examples for exporting a policy set, importing a policy set from an archive file, and importing a default policy set.

- Export an application or system policy set to an archive file.

  Use the exportPolicySet command to create an archive file for the policy set of interest. For example, the following command creates the customSC.zip archive file in the C:\IBM\WebSphere\AppServer\PolicySets\ directory for the customSecureConversation policy set:

  ```
  AdminTask.exportPolicySet('[-policySet customSecureConversation
  -pathName C:\IBM\WebSphere\AppServer\PolicySets\customSC.zip]')
  ```

- Move the policy set archive file to the destination environment.

  If you are exporting the policy set to a client environment, then place the archive file on the classpath of the client.

- Import a policy set from an archive file or import a default policy set.

  Use the importPolicySet command to import the archive file containing the policy set configuration of interest to the destination environment. You cannot import a policy set onto a server or client environment if the policy set already exists in the destination environment.

  For example, the following command creates a customSecureConversation policy set from the customSC.zip archive file:

  ```
  AdminTask.importPolicySet('[-importFile C:\IBM\WebSphere\AppServer\bin\customSC.zip]')
  ```

  Additionally, you can also use the importPolicySet command to import a default policy set onto a server environment, as the following example demonstrates:

  ```
  AdminTask.importPolicySet('[-defaultPolicySet SecureConversation -policySet copyOfdefaultSC]')
  ```

- Save the configuration changes.

  Enter the following command to save your changes:

  ```
  AdminConfig.save()
  ```

Removing policy set bindings using the wsadmin tool

You can use the Jython or Jacl scripting language to remove binding configurations for policies and resources to match your installation environment or requirements.

Before you begin

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete bindings. If you have access to a specific resource only, you can delete bindings for the resource for which you have access.
Configurator	The Configurator role can unassign bindings, but cannot delete bindings.
Deployer	The Deployer role can unassign bindings, but cannot delete bindings.
Operator	The Operator role cannot modify bindings.
Monitor	The Monitor role cannot modify bindings.

About this task

Use the following steps to remove specific policies from your application-specific binding configuration, or to remove your entire binding configuration. For both of these removal options, you must use the -bindingLocation parameter to specify whether you are deleting an application-specific binding, server-specific default binding, or a binding for the trust service. Use the following table for examples using Jython syntax when specifying the type of binding to modify or remove:

Type of binding	Value for the -bindingLocation parameter
Server-level (for Version 6.1 bindings only)	`-bindingLocation "[[node node1][server server1]]"`
Application	`-bindingLocation "[[application application1][attachmentId 123]]"`
Trust service binding	`-bindingLocation "[[systemType trustService] [attachmentId 123]]"`
WS-Notification client	`-bindingLocation "[[bus myBus][WSNService myService][attachmentId 123]"`
General bindings	`-bindingLocation []`

- Remove a policy from your application-specific binding configuration.

  Use the following steps to remove a specific policy from your binding configuration. If you remove the last policy remaining in your binding configuration, the command removes binding information from all attachments and deletes it from your configuration.

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Review the binding configuration to edit.

     Use the getBinding command to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 1234]]"')
```

     If the binding of interest is not referenced by an attachment ID, specify an asterisk character (*) for the attachmentId parameter to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId *]]"')
```

  3. Remove the policy from the binding configuration.

     Use the setBinding command with the -policyType and -remove parameters to remove the policy of interest from the binding configuration. For example, use the following command to remove the `WSAddressing` policy from the binding configuration for the `application1` application:

```
AdminTask.setBinding('-policyType WSAddressing -remove true -bindingLocation "[[application application1][attachmentId 1234]]"
```

     If the binding to delete is not referenced by an attachment ID, specify an asterisk character (*) for the attachmentId parameter to delete the binding, as the following example demonstrates:

```
AdminTask.setBinding('-policyType WSAddressing -remove true -bindingLocation "[[application application1][attachmentId *]]"')
```

4. Save your configuration changes.

- Remove binding configurations from an attachment.

  Use the following steps to remove a server-specific default binding or a custom binding. You cannot remove cell-level default bindings from your configuration. When a binding is removed from an attachment, the resource it was removed from will inherit the server-level default binding, if one is present, or the cell-level default binding if the server-level binding is not present. Use the following steps to remove a binding configuration:

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Verify the current binding configuration to delete.

     Before removing the binding from the attachment, use the getBinding command to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingLocation "[[application application1][attachmentId 123]]"')
```

  3. Remove the current binding configuration from the attachment.

     For this example, this command removes the bindings from the `123` attachment for the `application1` application:

```
AdminTask.setBinding('-bindingLocation "[[application application1][attachmentId 123]]" -remove true')
```

     If the binding to delete is not referenced by an attachment ID, specify an asterisk character (*) for the -attachmentId parameter to remove the binding, as the following example demonstrates:

```
AdminTask.setBinding('-bindingLocation "[[application application1][attachmentId *]]" -remove true')
```

     To remove a server-specific default binding, specify the node name and server name with the -bindingLocation parameter. Server specific default bindings are deprecated. For example, this command removes the server-level default binding for the WS-Addressing policy from the `server1` server on the `node1` node:

```
AdminTask.setBinding('-policyType WSAddressing -bindingLocation "[[node node1][server server1]]" -remove true')
```

  4. Save your configuration changes.

- Remove a policy from a general binding.

  Use the following steps to remove a

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Verify the current binding configuration to delete.

     Before removing the binding from the attachment, use the getBinding command to view the attributes for the binding, as the following example demonstrates:

```
AdminTask.getBinding('-policyType WSAddressing -bindingName "General Provider Binding" -bindingLocation []')
```

  3. Remove the general binding.

     For this example, this command removes the `General Provider Binding` general binding:

```
AdminTask.setBinding('-bindingLocation [] -bindingName "General Provider Binding" -remove true')
```

  4. Save your configuration changes.

Related tasks

“Reassigning bindings to policy sets” on page 856
After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

“Deleting application-specific bindings from your configuration using the wsadmin tool” on page 537
You can use the Jython or Jacl scripting language to delete a custom application or system policy set binding from your configuration. You cannot delete cell-level default bindings.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

“Removing policy set attachments using the wsadmin tool”
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Removing policy set attachments using the wsadmin tool

You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

Before you begin

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to remove policy set attachments. If you have access to a specific resource only, you can remove policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to remove policy set attachments. If you have access to a specific resource only, you can remove policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can remove policy set attachments for application resources only.
Operator	The Operator role cannot remove policy set attachments.
Monitor	The Monitor role cannot remove policy set attachments.

Determine which applications and policy sets to remove, detach, or transfer. Use the listWebServices command to list all Web services and for the application to edit. Enter the command to list all Web services and attributes for a specific application.

```
AdminTask.listWebServices('[–application application_name]')
```

To view a list of all Web services and associated applications, do not provide the -application parameter. For each Web service, the command returns the associated application name, module name, service name, and service type. You can also use the listAttachmentsForPolicySet and getPolicySetAttachments administrative commands to view existing configuration data. For additional information about these commands, use the information center topic for the PolicySetManagement group of commands for the AdminTask object.

About this task

There are four ways to remove policy set attachments, including:

- Remove a policy set attachment from an application.
- Remove resources that apply to a policy set attachment.
- Remove all attachments for a specific policy set and application.
- Transfer attachments between policy sets for a specific application.

Choose the appropriate procedure to remove your policy set attachments.

- Remove a policy set attachment from the application.
  1. Enter the following command to remove the policy set attachment from the application:

     ```
     AdminTask.deletePolicySetAttachment('[-attachmentId attachment_name -applicationName application_name]')
     ```

     The command returns a success or failure message. If you receive a success message, the policy set attachment is successfully removed from your application. If you receive a failure message, verify that the chosen policy set attachment exists in your configuration.
  2. Save the configuration changes.

     Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```
- Remove resources that apply to a policy set attachment.
  1. You can customize the command to remove resources for the Web service, endpoint, or operation.

     For the commands in the PolicySetManagement group, the term *resource* refers to a Web service artifact. For application and service client policy sets, the artifacts use the application hierarchy: Web service, module name, endpoint, or operation. Enter the value for the -resource parameter as a string, with a backslash (/) character as a delimiter. Use the following format for application and client policy set attachments:

WebService:/
Refers to all artifacts in the application to the policy set.

WebService:/webapp1.war:{http://www.ibm.com}myService
Refers to all artifacts within the Web service *{http://www.ibm.com}myService* to the policy set. You must provide a fully qualified name (QName) for the service.

WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA
Refers to all operations for the *endpointA* endpoint to the policy set.

WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1
Refers to only the *operation1* operation to the policy set.

The format for the **resource** string differs for system policy set attachments for the trust service. Use the following format for system policy set attachments:

Trust.<opName>:/
The *<opName>* attribute can be issue, renew, cancel, or validate.

Trust.<opName>:/url
The *<opName>* attribute can be issue, renew, cancel, or validate. You can specify any valid URL for the *url* attribute.

In the following example, the command removes the `attachment_name` attachment from the `operation1` operation, which is associated with the application, `application1`.

```
AdminTask.removeFromPolicySetAttachment('[-attachmentId attachment_name -resources
"WebService:/webapp1.war:{http://www.ibm.com}myService/endpointA/operation1" -applicationName application1]')
```

The command returns a success or failure message. You can also use the **updatePolicySetAttachments** command to remove attached resources.

2. Save the configuration changes.

   Enter the following command to save your changes:

   ```
   AdminConfig.save()
   ```

- Remove all attachments for a specific policy set.
  1. Remove application attachments for the policy set. To remove an attachment from an application, use the following command:

     ```
     AdminTask.deleteAttachmentsForPolicySet('[-policySet PolicySet1 -applicationName application1]')
     ```

     To remove all attachments from the policy set, use the following command:

     ```
     AdminTask.deleteAttachmentsForPolicySet('[-policySet PolicySet1]')
     ```

     Both commands return a success or failure message.
  2. Save the configuration changes.

     Enter the following command to save your changes:

     ```
     AdminConfig.save()
     ```

- Transfer attachments from one policy set to another policy set. This command detaches all Web services from the source policy set and attaches the Web services to the destination policy set.
  1. Enter the transferAttachmentsForPolicySet command to transfer all attachments within an application. Use the following command to transfer the attachments from the *PolicySet1* policy set to the *PolicySet2* policy set within the *application1* application:

     ```
     AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet PolicySet1
     -destinationPolicySet PolicySet2 -applicationName application1]')
     ```

     The command returns a success or failure message.
  2. Save the configuration changes.

     Enter this command to save your changes:

     ```
     AdminConfig.save()
     ```

Related tasks

Configuring attachments for the trust service using the administrative console
You can attach the trust service operations for a service endpoint to a system policy set and binding. Each new endpoint that is specified initially has the following four operations: issue, renew, cancel, and validate. By default, all endpoints inherit the policy set and binding that are attached to the respective trust service operation under Trust Service Defaults. However, you can explicitly attach a different policy set.

"Configuring application and system policy sets for Web services using scripting" on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

"Creating policy sets using the wsadmin tool" on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

"Adding and removing policies using the wsadmin tool" on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

"Creating policy set attachments using the wsadmin tool" on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

"Managing policy set attachments using the wsadmin tool" on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

"Removing policy set attachments using the wsadmin tool" on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Related reference

"PolicySetManagement command group for the AdminTask object" on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Deleting policy sets using the wsadmin tool

Use the Jython or Jacl scripting language to delete policy sets from your configuration with the wsadmin tool. You must remove all policy set attachments before removing the policy set.

Before you begin

To complete this task, you must use the Administrator role with cell-wide access when administrative security is enabled.

Before deleting a policy set, you must delete or transfer all attachments to applications. You cannot delete default policy sets.

About this task

Use the following steps to delete custom policy sets from your configuration with the wsadmin tool:

1. Launch a scripting command.
2. List all policy sets in your configuration.
   - Enter the following command to list all application policy sets:
     ```
     AdminTask.listPolicySets()
     ```
   - Enter the following command to list all policy sets for the trust service:
     ```
     AdminTask.listPolicySets('[-policySetType system/trust]')
     ```
   - Enter the following command to list all system policy sets:
     ```
     AdminTask.listPolicySets('[-policySetType system]')
     ```
3. Determine which policy set to delete. Enter the following command to view the description and default indicator for a specific policy set:
   ```
   AdminTask.getPolicySet('[-policySet policySet_name]')
   ```
4. Delete the policy set.

   Enter the following command to delete a specific policy set.
   ```
   AdminTask.deletePolicySet('[-policySet PolicySet1)')
   ```

   The command returns a success or failure response. If you receive an error message, make sure that you have deleted all policy set attachments before entering the deletePolicySet command.
5. Save the configuration changes.

   Enter the command to save your changes.
   ```
   AdminConfig.save()
   ```

What to do next

If you deleted a policy set that was previously attached to an application, restart the affected application to update the configuration changes.

Related tasks

“Deleting policy sets using the administrative console” on page 822
You can use the administrative console to delete the default policy sets or the application specific policy sets that you have created.

“Configuring application and system policy sets for Web services using scripting” on page 497
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to configure application or system policy sets for Web services. You can manage the policies for the Quality of Service (QoS) by creating policy sets and managing associated policies.

“Creating policy sets using the wsadmin tool” on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

“Adding and removing policies using the wsadmin tool” on page 503
You can use the Jython or Jacl scripting language and the wsadmin tool to query, add, and remove policies for your policy sets.

Related reference

“PolicySetManagement command group for the AdminTask object” on page 573
You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Refreshing policy set configurations using scripting

Use the wsadmin tool to refresh the policy set configuration data. After refreshing the policy set configuration, the changes apply after restarting the application.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Get the object name of each PolicySetManager object.

   Use the completeObjectName option for the AdminControl object to set the object name for each PolicySetManager type object to the `objNameString` variable, as the following example demonstrates:

```
objNameString = AdminControl.completeObjectName('type=PolicySetManager,*')
```

3. Connect to the Managed Bean (MBean).

   The MBean supplies a remote interface to the MBean server that runs in the application server. The following example shows how to look up the MBean:

```
import javax.management as mgmt
```

4. Set the PolicySetManager MBean object name.

   The following example sets the PolicySetManager MBean object name to the `mbeanObj` variable, parameters to the `param` variable, and signature settings to the `sig` variable:

```
mbeanObj = mgmt.ObjectName(objNameString)
param=[]
sig=[]
```

5. Refresh the PolicySetManager MBean.

   The following example refreshes the policy set configuration:

```
AdminControl.invoke_jmx(mbeanObj, 'refresh', param, sig)
```

Example

The following example provides the Jython script that refreshes the policy set configuration:

```
objNameString = AdminControl.completeObjectName('type=PolicySetManager,*')
import javax.management as mgmt
mbeanObj = mgmt.ObjectName(objNameString)
param=[]
sig=[]
AdminControl.invoke_jmx(mbeanObj, 'refresh', param, sig)
```

Policy configuration properties for all policies

You can use the **attributes** parameter with the setPolicyType and setBinding commands to specify various properties for each quality of service (QoS) within a policy set. You can use the properties in this topic with each QoS within application and system policy sets.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Attributes to configure for all QoS policies

Use the following list of attributes to configure attributes across all QoS policies using the Jython scripting language and the wsadmin tool:

enabled

Specifies whether the policy type is enabled or disabled. The following example provides the format to enter the attributes parameter:

```
-attributes "[[enabled true]]"
```

provides

Provides a description for your configuration. The following example provides the format to enter the attributes parameter:

```
-attributes "[[provides [Messaging Security]]]"
```

The following example uses the setPolicyType command to set the `enabled` and `provides` properties for the myCustomSecurityPS custom policy set, which contains a ReliableMessaging policy:

```
AdminTask.setPolicyType('[-policySet myCustomSecurityPS -policyType
WSReliableMessaging -attributes [[enabled true][provides
[Messaging security]]]]')
```

WSSecurity policy and binding properties

Use the **attributes** parameter for the setPolicyType and setBinding commands to specify additional configuration information for the WSSecurity policy and binding configurations. Application and system policy sets can use the WSSecurity policy and binding configuration.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (″″). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

If the **attributes** parameter is not specified for the getPolicyType or getBinding command, the command returns all properties. If a partial property name is passed to the getPolicyType or getBinding command, the command returns all properties with names that start with the partial property name. For example, If SignatureProtection is passed to the getPolicyType command, the command returns all properties with names that start with ″SignatureProtection″, which might include:

```
SignatureProtection.response:
   int_body.SignedParts.Body,SignatureProtection.response:int_body.SignedParts.Header_0.Name
```

, and

```
SignatureProtection.response:int_body.SignedParts.Header_0.Namespace
```

.

There are an extensive number of combinations of settings that are available to secure your Web service applications. Because of the number of attributes and configuration options from the WS-Security Version 1.0 specification, all attributes are not defined in this topic. The following sections explain the hierarchy structure for the WSSecurity policy and binding attributes:

- WSSecurity policy properties
- WSSecurity binding properties
- “setPolicyType and setBinding command examples” on page 556

WSSecurity policy properties

Use the getPolicyType command to review a properties object with the properties that are configured in your current WSSecurity policy file. Security policy schemata define the security assertions. Because the elements in the schema have hierarchical relationship, the property names for security policy also have the similar hierarchy. The hierarchical relationship between property names in the security policy is represented by a period (.) between two levels, concatenating the parent and child attributes. Examples of the properties include, but are not limited to, IncludeToken, Name, Namespace, XPath, XPathVersion. The following list describes the top-level assertion policy property names for the WSSecurity policy file:

AsymmetricBinding
You can specify zero or one binding assertion.

SymmetricBinding
You can specify zero or one binding assertion. AsymmetricBinding and SymmetricBinding cannot co-exist in a security policy file.

Wss11
You can specify zero or one Wss11 assertion.

Wss10
You can specify zero or one Wss10 assertion.

Trust10
You can specify zero or one Trust10 assertion.

SignatureProtection
You can specify zero or any number of signature protection assertions.

EncryptionProtection
You can specify zero or any number of encryption protection assertions

SupportingTokens
You can specify zero or any number of supporting token assertions.

For example, the following policy file example displays an AsymmetricBinding assertion:

```
<sp:AsymmetricBinding>
  <wsp:Policy>
    <sp:InitiatorSignatureToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
  /200512/IncludeToken/AlwaysToRecipient">
          <wsp:Policy>
            <sp:WssX509V3Token10 />
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorSignatureToken>
    <sp:RecipientSignatureToken>
      <wsp:Policy>
        <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
  /200512/IncludeToken/AlwaysToInitiator">
          <wsp:Policy>
            <sp:WssX509V3Token10 />
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientSignatureToken>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
```

```
      </sp:Layout>
    </wsp:Policy>
  </sp:AsymmetricBinding><sp:AsymmetricBinding>
    <wsp:Policy>
      <sp:InitiatorSignatureToken>
        <wsp:Policy>
          <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
    /200512/IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
              <sp:WssX509V3Token10 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:InitiatorSignatureToken>
      <sp:RecipientSignatureToken>
        <wsp:Policy>
          <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy
    /200512/IncludeToken/AlwaysToInitiator">
            <wsp:Policy>
              <sp:WssX509V3Token10 />
            </wsp:Policy>
          </sp:X509Token>
        </wsp:Policy>
      </sp:RecipientSignatureToken>
    </wsp:Policy>
    <sp:AlgorithmSuite>
      <wsp:Policy>
        <sp:Basic256/>
      </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
      <wsp:Policy>
        <sp:Strict/>
      </wsp:Policy>
    </sp:Layout>
  </sp:AsymmetricBinding>
```

The AsymmetricBinding assertion returns the following property name and value pairs. The nested `wsp:Policy` layers are not displayed in the returned properties. Additionally, some properties return the `true` value which indicates that the WSSecurity configuration includes the related XML elements. To edit these properties, set the value as `true` to include the property, or set the value as an empty string,"", to remove the property.

```
AsymmetricBinding.Layout = Strict
AsymmetricBinding.AlgorithmSuite.Basic256 = true
AsymmetricBinding.RecipientSignatureToken.X509Token_0.IncludeToken = http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToInitiator
AsymmetricBinding.InitiatorSignatureToken.X509Token_0.WssX509V3Token10 = true
AsymmetricBinding.InitiatorSignatureToken.X509Token_0.IncludeToken = http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient
AsymmetricBinding.RecipientSignatureToken.X509Token_0.WssX509V3Token10 = true
```

Additionally, the following policy file example displays a SupportingTokens assertion:

```
<sp:SupportingTokens>
        <wsp:Policy wsu:Id="request:custom_auth">
            <spe:CustomToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/
       ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient">
                <wsp:Policy>
                    <spe:WssCustomToken uri=http://bar.com/MyCustomToken localname="tokenv1">
                    </spe:WssCustomToken>
                </wsp:Policy>
            </spe:CustomToken>
        </wsp:Policy>
    </sp:SupportingTokens
```

The SupportingTokens assertion returns the following property name and value pairs. The nested `wsp:Policy` layers are not displayed in the returned property.

```
SupportingTokens.request:custom_auth.CustomToken_0.WssCustomToken.uri=http://bar.com
/MyCustomToken
SupportingTokens.request:custom_auth.CustomToken_0.IncludeToken=http://docs.oasis-open.org
/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient
SupportingTokens.request:custom_auth.CustomToken_0.WssCustomToken.localname=tokenv1
```

Note: The CustomToken property contains a subscript zero notation (_0) because the property might be displayed multiple times from the same type of token such as the RecipientSignatureToken or InitiatorSignatureToken tokens.

Although most property names follow the hierarchical relationship format described previously, the following exceptions exist:

- The wsu:Id element

  This element uses the actual value for the ID instead of using `Id` as the attribute name. The following policy file example property:

  ```
  <wsp:Policy wsu:Id="response:int_body">
    <sp:SignedParts>
      <sp:Body/>
    </sp:SignedParts>
  </wsp:Policy>
  ```

  The previous wsu:Id example returns the following properties:

  ```
  SignatureProtection.response:int_body.SignedParts.Body = true
  ```

- The Header element

  Because there can be multiple Header elements, the `Header_n` notation is used to represent this property. See the following policy file example:

  ```
  <wsp:Policy wsu:Id="request:conf_body">
     <sp:EncryptedParts>
       <sp:Body/>
       <sp:Header Name="MyElement" Namespace="http://foo.com/MyNamespace" />
     </sp:EncryptedParts>
   </wsp:Policy>
  ```

  The previous Header example returns the following properties:

  ```
  EncryptionProtection.request:conf_body.EncryptedParts.Header_0.Name=MyElement
  EncryptionProtection.request:conf_body.EncryptedParts.Header_0.Namespace=http://
  foo.com/MyNamespace
  ```

- The XPath element

  The `XPath_n` notation is used to represent this property because there can be multiple XPath elements. See the following policy file example:

  ```
  <wsp:Policy wsu:Id="request:int_body">
       <sp:SignedElements>
         <sp:XPath>SomeXPathExpression</sp:XPath>
         <sp:XPath>SomeOtherXPathExpression</sp:XPath>
       </sp:EncryptedElements>
     </wsp:Policy>
  ```

  The previous XPath example returns the following properties:

  ```
  SignatureProtection.request:int_body.SignedElements.XPath_0=SomeXPathExpression
  SignatureProtection.request:int_body.SignedElements.XPath_1=SomeOtherXPathExpression
  ```

- The X509Token element

  Use the X509Token_*n* notation to represent this property because multiple X509Token elements can exist. For an example, see the AsymmetricBinding assertion.

- The CustomToken element

  Use the CustomToken_*n* notation to represent this property because multiple CustomToken elements can exist. For an example, see the SupportingTokens assertion.

WSSecurity binding properties

Use the getBinding command to review a properties object with the properties that are configured in your current WSSecurity binding configuration. You can also use the administrative console to configure your WSSecurity bindings. Use the information center topics for configuring WSSecurity bindings with administrative console for more information.

The properties defined in this section reflect the hierarchy of the binding schema. Each part of the property name is a lowercase version of the schema type. For example, the application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname property follows the hierarchal format. The attributes begin with **application** or **bootstrap**. Attributes that begin with

application represent bindings that are associated with the main WS-Security policy. Attributes that begin with **bootstrap** represent bindings that are associated with the WS-Security bootstrap policy, where the WS-Security policy uses Secure Conversation.

Some property names might have an _n notation appended to them. This notation represents a list of items. For example, multiple **tokenconsumer** properties exist and are listed from tokenconsumer_*0* through tokenconsumer_*n*, where the set of **tokenconsumer** values are:

```
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.
certpathsettings.certstoreref.reference
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.
certpathsettings.trustanchorref.reference
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname
application.securityinboundbindingconfig.tokenconsumer_0.classname
application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname
application.securityinboundbindingconfig.tokenconsumer_0.name
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri
```

Additionally, some properties in the security binding file return a value of true when queried. To set these properties, set the value to true to include the property, or set the value to an empty string ("") to remove the property. For example, the time stamp, nonce, and trustAnyCertificate properties follow this pattern.

Use the setBinding command and the **attributes** parameter to add or remove properties to your WSSecurity binding configuration.

- To add a property, use the setBinding command to pass the property name with a non-zero length string value. To add a list item, use the _n notation to reflect a numeric value that is greater than any current numeric value for the property. For example, if the tokenconsumer_0 and tokenconsumer_1 properties exist in your configuration, specify the new tokenconsumer property as tokenconsumer_2. After adding a property, use the getBinding command to view the most recent list of configured properties.
- To remove a property, use the setBinding command to pass the property name with an empty string (""). For example, to remove all of the tokenconsumer_0 properties, specify the following property with the **attributes** parameter:

  ```
  application.securityinboundbindingconfig.tokenconsumer_0=""
  ```

  The previous example removes all properties that begin with the application.securityinboundbindingconfig.tokenconsumer_0 property name.

The following examples display several sets of properties to configure for your binding. This list does not include all properties to configure for the WSSecurity binding. Use this information as a reference to determine how to form specific property names.

signinginfo element

Use this property to configure signing information. For a custom binding, an unlimited number of **signinginfo** elements specified for the securityoutboundbindingconfig and securityinboundbindingconfig assertions can exist. In the default bindings, the system allows a maximum of two **signinginfo** elements for the securityoutboundbindingconfig and securityinboundbindingconfig assertions. The following example displays the format for two **signinginfo** elements:

```
application.securityinboundbindingconfig.signinginfo_0.signingkeyinfo_0
.reference=con_signkeyinfo
application.securityinboundbindingconfig.signinginfo_0.signingpartreference_0
.reference=request:int_body
application.securityoutboundbindingconfig.signinginfo_0.signingpartreference_0
.reference=response:int_body
application.securityoutboundbindingconfig.signinginfo_0.signingpartreference_0.timestamp=true
```

encryptioninfo element

Use this property to configure encryption information. For a custom binding, an unlimited number of **encryptioninfo** elements specified for the securityoutboundbindingconfig and securityinboundbindingconfig assertions can exist. In the default bindings, the system accepts a

maximum of two **encryptioninfo** elements for the `securityoutboundbindingconfig` and `securityinboundbindingconfig` assertions. The following example displays the format for two **encryptioninfo** properties:

```
application.securityinboundbindingconfig.encryptioninfo_0.encryptionpartreference
.nonce=true
application.securityinboundbindingconfig.encryptioninfo_0.encryptionpartreference
.reference=request:conf_body
application.securityoutboundbindingconfig.encryptioninfo_0.encryptionpartreference
.nonce=true
application.securityoutboundbindingconfig.encryptioninfo_0.encryptionpartreference
.timestamp=true
```

tokengenerator element

In the default bindings, the **tokengenerator** elements that the **signinginfo** or **encryptioninfo** elements do not reference are considered to be authentication token generators. Each authentication token generator must have a unique **valuetype** element. The following example displays an example of a generator for an X.509 protection token:

```
application.securityoutboundbindingconfig.tokengenerator_0.name=gen_signtgen
application.securityoutboundbindingconfig.tokengenerator_0.classname=com.ibm.ws.wssecurity.wssapi.token
.impl.CommonTokenGenerator
application.securityoutboundbindingconfig.tokengenerator_0.valuetype.uri=
application.securityoutboundbindingconfig.tokengenerator_0.valuetype.localname=http://docs.oasis-open.org
/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.classname=com.ibm.websphere.wssecurity
.callbackhandler.X509GenerateCallbackHandler
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.alias=soaprequester
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.keypass={xor}PDM2OjEr
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.key.name=CN=SOAPRequester,
OU=TRL, O=IBM, ST=Kanagawa, C=JP
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.path=${USER_INSTALL_ROOT}
/etc/ws-security/samples/dsig-sender.ks
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.storepass={xor}PDM2OjEr
application.securityoutboundbindingconfig.tokengenerator_0.callbackhandler.keystore.type=JKS
application.securityoutboundbindingconfig.tokengenerator_0.jaasconfig.configname=system.wss.generate.x509
```

The following example displays a generator for a username authentication token:

```
application.securityoutboundbindingconfig.tokengenerator_1.name=gen_usernametoken
application.securityoutboundbindingconfig.tokengenerator_1.classname=com.ibm.ws.wssecurity
.wssapi.token.impl.CommonTokenGenerator
application.securityoutboundbindingconfig.tokengenerator_1.valuetype.uri=
application.securityoutboundbindingconfig.tokengenerator_1.valuetype.localname=http://docs
.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.classname=com.ibm
.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.basicAuth.userid=user1
application.securityoutboundbindingconfig.tokengenerator_1.callbackhandler.basicAuth.password=myPassword
application.securityoutboundbindingconfig.tokengenerator_1.securityTokenReference.reference=request:uname_token
application.securityoutboundbindingconfig.tokengenerator_1.jaasconfig.configname=system.wss.generate.unt
```

tokenconsumer element

In the default bindings, the **tokenconsumer** elements that the **signinginfo** or **encryptioninfo** elements do not reference are authentication token consumers. Each authentication token consumer must have a unique **valuetype** element. The following example displays the format for a set of **tokenconsumer** elements:

```
application.securityinboundbindingconfig.tokenconsumer_0.name=con_unametoken
application.securityinboundbindingconfig.tokenconsumer_0.classname=com.ibm.ws.wssecurity.wssapi
.token.impl.CommonTokenConsumer
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.localname=http://docs.oasis-open.org
/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken
application.securityinboundbindingconfig.tokenconsumer_0.valuetype.uri=
application.securityinboundbindingconfig.tokenconsumer_0.callbackhandler.classname=com.ibm.websphere
.wssecurity.callbackhandler.UNTConsumeCallbackHandler
application.securityinboundbindingconfig.tokenconsumer_0.jaasconfig.configname=system.wss.consume.unt
application.securityinboundbindingconfig.tokenconsumer_0.securitytokenreference.reference=request:uname_token
```

actor element

Defines the actor uniform resource identifier (URI) to be included in the WSSecurity headers of a generated message, as displayed by the following example:

```
application.securityinboundbindingconfig.actor=http://myActor.com
application.securityoutboundbindingconfig.actor=http://myActor.com
```

certstorelist element

Defines certificate store configurations and signing information, as displayed by the following example:

```
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.name=DigSigCertStore
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.provider=IBMCertPath
application.securityinboundbindingconfig.certstorelist.collectioncertstores_0
.x509certificates_0.path=${USER_INSTALL_ROOT}/etc/ws-security/samples/intca2.cer
```

keyinfo element

Defines key information for signing and encryption configurations, as displayed by the following example:

```
application.securityinboundbindingconfig.keyinfo_0.classname=com.ibm.ws.wssecurity.wssapi
.CommonContentConsumer
application.securityinboundbindingconfig.keyinfo_0.name=con_signkeyinfo
application.securityinboundbindingconfig.keyinfo_0.tokenreference.reference=con_tcon
application.securityinboundbindingconfig.keyinfo_0.type=STRREF
```

trustanchor property

Defines configuration information that is used to validate the trust of the signer certificate, as displayed by the following example:

```
application.securityinboundbindingconfig.trustanchor_0.keystore.path=${USER_INSTALL_ROOT}
/etc/ws-security/samples/dsig-receiver.ks
application.securityinboundbindingconfig.trustanchor_0.keystore.storepass={xor}LDotKTot
application.securityinboundbindingconfig.trustanchor_0.keystore.type=JKS
application.securityinboundbindingconfig.trustanchor_0.name=DigSigTrustAnchor
```

timestampexpires element

Defines an expiration date for the configuration, as displayed by the following example:

```
application.securityoutboundbindingconfig.timestampexpires.expires=5
```

application.securityinboundbindingconfig.caller_X.order

Specifies the order for a caller when using wsadmin scripts, where X is the unique string that identifies the instance of the caller:

```
-attributes [[application.securityinboundbindingconfig.caller_0.order 2]]
```

setPolicyType and setBinding command examples

Use the previous reference information with the setPolicyType and setBinding commands to modify your policy and binding configuration data.

Note: The administrative console command assistance provides incorrect Jython syntax for the setPolicyType command. The XPath expression for the response message part protection of the Username WSSecurity policy set contains single quotes (') within each XPath property value, which Jython does not support. To fix the command from the administrative console command assistance, add a backslash character (\) before each single quote to escape the single quote.

The following example uses the setBinding command to set the enabled and provides properties for the myCustomSecurityPS custom policy set, which contains a ReliableMessaging policy:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType WSSecurity
 -attributes [[application.securityinboundbindingconfig.caller_0.order 2][inResponsewithSSL:configAlias NodeDefaultSSLSettings]
[inResponsewithSSL:config properties_directory/ssl.client.props][outAsyncResponsewithSSL:configFile properties_directory/ssl.client.props]
[outAsyncResponsewithSSL:configAlias NodeDefaultSSLSetings][outRequestwithSSL:configFile properties_directory/ssl.client.props]
[outRequestwithSSL:configAlias NodeDefaultSSLSettings]]]')
```

The following setPolicyType command enables the WSSecurity policy and creates a signature protection assertion:

```
AdminTask.setPolicyType('-policySet myPolicySet -policyType WSSecurity -attributes "[[enabled true][provides
Some_amount_of_security][SignatureProtection.request:app_signparts.SignedElements.XPath_0 SignatureProtectionV2]]"')
```

The following setBinding command specifies key information for a server-specific binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[[server server1][node node01]]"
-attributes "[[application.securityinboundbindingconfig.keyinfo_0.name dec_server_keyinfo]
[application.securityinboundbindingconfig.keyinfo_0.classname com.ibm.ws.wssecurity.wssapi.CommonContentGenerator]
[application.securityinboundbindingconfig.keyinfo_0.type STRREF]]"')
```

The following setBinding command specifies key information for an attachment-specific binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "[[application PolicySet][attachmentId 999]]"
-attributes "[[application.securityinboundbindingconfig.keyinfo_0.name dec_app_keyinfo]
[application.securityinboundbindingconfig.keyinfo_0.classname com.ibm.ws.wssecurity.wssapi.CommonContentGenerator]
[application.securityinboundbindingconfig.keyinfo_0.type STRREF]]" -attachmentType application
 -bindingName myBindingName')
```

The following setBinding command specifies trust anchor information for a cell-wide binding:

```
AdminTask.setBinding('-policyType WSSecurity -bindingLocation "" -attributes
"[application.securityinboundbindingconfig.trustanchor_0.name DigSigTrustAnchor2]"')
```

WSReliableMessaging policy and binding properties

Use the **attributes** parameter for the setPolicyType and setBinding commands to specify additional configuration information for the ReliableMessaging policy and policy set binding. The WSReliableMessaging quality of service (QoS) is only available for application policy sets.

WSReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. Use WSReliableMessaging to secure and verify transactions when using Web services between businesses.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

WSReliableMessaging policy properties

Configure the WSReliableMessaging policy by specifying the following properties with the setPolicyType command:

specLevel

Choose the WS-ReliableMessaging standard to use for reliable transmission of your messages. The WS-ReliableMessaging specification Version 1.1 is the default value. Use the following information to choose a specification level:

- Specify `1.0` as the value for the specLevel attribute to use the WS-ReliableMessaging specification Version 1.0, February 2005 specification level.
- Specify `1.1` as the value for the specLevel attribute to use the OASIS WS-ReliableMessaging specification Version 1.1, August 2006 specification level.

The following example code sets the specLevel property to the OASIS WS-ReliableMessaging specification Version 1.1, August 2006:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging"  -policyType
 WSReliableMessaging -attributes "[[specLevel 1.1]]"]')
```

inOrderDelivery

Specifies whether to process messages in the order that they are received. If you use the inOrderDelivery property, then inbound messages might be queued while waiting for earlier messages.

The following example code enables the inOrderDelivery property:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType WSReliableMessaging -attributes "[[inOrderDelivery true]]"]')
```

qualityOfService

Specifies the quality of the WSReliableMessaging service to use. Define one of the following three values for the qualityOfService attribute:

- unmanagedNonPersistent

  This setting tolerates network and remote system failures. The unmanagedNonPersistent quality of service is non-transactional. With this setting configured, messages are lost if a server fails. This quality of service is supported for all environments only if the environment is configured as a Web service requester.
- managedNonPersistent

  This setting tolerates system, network, and remote system failures. However, the message state is discarded when the messaging engine restarts. The managedNonPersistent quality of service is non-transactional. This setting prevents message loss if a server fails. However, messages are lost if the messaging engine fails. Managed and thin client applications cannot use this quality of service.
- managedPersistent

  This setting tolerates system, network, and remote system failures. With this setting, messages are processed within transactions, persisted at the Web service requester and provider. Messages can be recovered if a server fails. Messages that are not successfully transmitted at the time of failure continue when the messaging engine or application restarts. Managed and thin client applications cannot use this quality of service.

The following example sets the qualityOfService property as unmanaged nonpersistent:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType
 WSReliableMessaging -attributes "[[qualityOfService unmanagedNonPersistent]]"]')
```

The following example uses the setPolicyType command to set a value for each policy property:

```
AdminTask.setPolicyType('[-policySet "CustomWSReliableMessaging" -policyType
 WSReliableMessaging -attributes "[[specLevel 1.1][inOrderDelivery true][qualityOfService
 unmanagedNonPersistent]]"]')
```

WSReliableMessaging binding configuration attributes

If you set the qualityOfService policy property to managedNonPersistent or managedPersistent, configure the WSReliableMessaging binding by specifying values for the following properties with the setBinding command:

busName

The name of the service integration bus that contains the messaging engine to use for the managedNonPersistent or managedPersistent Quality of Service options.

The following example sets the busName property as `myBus`:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[busName myBus]]"]')
```

messagingEngineName

The name of the messaging engine to use for the managedNonPersistent or managedPersistent quality of service options.

The following example sets the messagingEngineName property as `messagingEngine001`:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[messageEngineName messageEngine001]]"]')
```

The following code example demonstrates how to use the **setBinding** command to set values for each binding attribute:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 WSReliableMessaging -attributes "[[busName myBus][messageEngineName messageEngine001]]"]')
```

WSAddressing binding properties

Use the -attributes parameter for the setBinding command to enable or disable workload management for the WSAddressing binding. Application and system policy sets use the WSAddressing policy and binding.

WSAddressing is an interoperability standard that you can use to create endpoint references that you can distribute across firewalls and intermediary nodes. For more information, see the W3C Candidate Recommendation (CR) versions of the WS-Addressing core and SOAP specifications.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

WSAddressing binding properties

Configure the WSAddressing policy by specifying the following property with the setBinding command:

preventWLM
Specifies whether to prevent workload management for references to endpoints that were created by the application programming interface (API) in a cluster environment. Messages that target Endpoint References (EPRs) within a cluster environment are workload managed by default.

Preventing workload management routes messages that target EPRs to the node or server on which the EPR was created. You might disable workload management if the endpoint maintains the in-memory state, which has not been replicated across other nodes or servers within the cluster.
For example, the following command prevents workload management for a cell-wide general binding, from the WSAddressing policy.

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType WSAddressing -attributes "[preventWLM true]"]')
```

SSLTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the SSLTransport policy and policy set binding. Application and system policy sets can use the SSLTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the SystemOut.log file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the SystemOut.log file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

SSLTransport policy properties

Use the SSLTransport policy to ensure message security.

Configure the SSLTransport policy by specifying the following properties with the setPolicyType command:

outRequestSSLenabled
Specifies whether to enable the SSL security transport for outbound service requests.

outAsyncResponseSSLenabled
Specifies whether to enable the SSL security transport for asynchronous service responses.

inResponseSSLenabled
Specifies whether to enable the SSL security transport for inbound service responses.

The following setPolicyType command example sets values for all SSLTransport policy properties:

```
AdminTask.setPolicyType('[-policySet "WSHTTPS default" -policyType SSLTransport
-attributes "[[inReponseSSLenabled yes][outAsyncResponseSSLenabled yes][outRequestSSLenabled
 yes]]"]')
```

SSLTransport binding properties

Use the SSLTransport policy type to ensure message security.

Configure the SSLTransport binding by specifying the following properties using the setBinding command:

outRequestwithSSL:configFile
outRequestwithSSL:configAlias
If you enable SSL outbound service requests, then these two attributes define the specific SSL security transport binding and location. The default value for the outRequestwithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the outRequestwithSSL:configAlias attribute is `NodeDefaultSSLSettings`.

outAsyncResponsewithSSL:configFile
outAsyncResponsewithSSL:configAlias
If you enable SSL asynchronous service responses, then these two attributes define the specific SSL security transport binding and location. The default value for the outAsyncRequestwithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the outAsyncRequestwithSSL:configAlias attribute is `NodeDefaultSSLSettings`.

inResponsewithSSL:configFile
inResponsewithSSL:configAlias
If you enable SSL inbound service responses, then these two attributes define the specific SSL security transport binding and location. The default value for the inResponsewithSSL:configFile attribute is the location of the ssl.client.props file. The default value for the inResponsewithSSL:configAlias property is `NodeDefaultSSLSettings`.

The following setBinding command example sets values for all SSLTransport binding attributes:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName cellWideBinding2 -policyType
 SSLTransport -attributes "[[inResponsewithSSL:configAlias NodeDefaultSSLSettings] [inResponsewithSSL:config
 properties_directory/ssl.client.props][outAsyncResponsewithSSL:configFile properties_directory/ssl.client.props]
[outAsyncResponsewithSSL:configAlias NodeDefaultSSLSetings][outRequestwithSSL:configFile
 properties_directory/ssl.client.props][outRequestwithSSL:configAlias NodeDefaultSSLSettings]]"]')
```

HTTPTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the HTTPTransport policy and policy set binding. Application and system policy sets can use the HTTPTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

The following sections explain the policy and binding properties to configure:

- HTTPTransport policy properties
- HTTPTransport binding properties

HTTPTransport policy properties

The HTTPTransport policy set can be used for HTTPS, basic authorization, compression, and binary encoding transport methods.

Configure the HTTPTransport policy by specifying the following attributes with the **setPolicyType** command:

protocolVersion
Specifies the version of HTTP to use. The valid version values are `HTTP/1.1` and `HTTP/1.0`.

maintainSession
Specifies whether the HTTP session is enabled when a message is sent. The valid values are `yes` or `no`.

chunkTransferEnc
Specifies whether to enable chunked transfer encoding. The valid values are `yes` or `no`.

sendExpectHeader
Specifies whether to send an expect 100-request header. The valid values are `yes` or `no`.

compressRequest:name
Specifies whether to compress the request. The valid values are `gzip`, `x-gzip`, `deflate`, or `none`.

compressResponse:name
Specifies whether to compress the response. The valid values are `gzip`, `x-gzip`, `deflate`, or `none`.

acceptRedirectionURL
Specifies whether to accept URL redirection automatically. The valid values are `yes` or `no`.

messageResendOnce
Specifies if a message can be sent more than once. The valid values are `yes` or `no`.

connectTimeout
Specifies the amount of time, in seconds, before a connection times out when sending a message. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 180 seconds. No maximum value is set for this property.

writeTimeout
Specifies the amount of time, in seconds, before the write time out occurs. Specify an integer value. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 300 seconds. No maximum value is set for this property.

readTimeout
Specifies the amount of time, in seconds, before the read time out occurs. Specify an integer value. Specify an integer value that is greater than zero. If a value of zero or less is specified, the connectTimeout property is set to the default value of 300 seconds. No maximum value is set for this property.

persistConnection
Specifies whether to use a persistent connection when sending messages. Valid values are `yes` or `no`.

The following setPolicyType example command sets values for each HTTPTransport binding property:

```
AdminTask.setPolicyType('[-policySet "WSHTTPS custom" -policyType HTTPTransport -attributes "[[protocolVersion HTTP/1.1]
[sessionEnable yes][chunkTransferEnc yes][sendExpectHeader yes][compressRequest:name gzip][compressResponse:name
 gzip][acceptRedirectionURL yes][messageResendOnce no][connectTimeout 300][writeTimeout 300]
[readTimeout 300][persistConnection yes]]"]')
```

HTTPTransport binding properties

Configure the HTTPTransport binding by specifying the following attributes with the **setBinding** command:

outAsyncResponseBasicAuth:userid
Specifies the user name for basic authentication of outbound asynchronous responses.

outAsyncResponseBasicAuth:password
Specifies the password for basic authentication of outbound asynchronous responses.

outAsyncResponseProxy:userid
Specifies the user name for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:password
Specifies the password for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:port
Specifies the port number for the outbound asynchronous service responses proxy.

outAsyncResponseProxy:host
Specifies the host name for the outbound asynchronous service responses proxy.

outRequestBasicAuth:userid
Specifies the user name or basic authentication of outbound service requests.

outRequestBasicAuth:password
Specifies the password for basic authentication of outbound service requests.

outRequestProxy:userid
Specifies the user name for the outbound service request proxy.

outRequestProxy:password
Specifies the password for the outbound service request proxy.

outRequestProxy:port
Specifies the port number for the outbound service request proxy.

outRequestProxy:host
Specifies the host name for the outbound service request proxy.

The following **setBinding** example command sets values for each HTTPTransport binding property:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName generalCellWideBind1 -policyType HTTPTransport
-attributes "[[outAsyncResponseBasicAuth:userid myID][outAsyncResponseBasicAuth:password myPW][outAsyncResponseProxy:host hostname]
[outAsyncResponseProxy:port 9060][outAsyncResponseProxy:userid myID][outAsyncResponseProxy:password myPW]
[outRequestBasicAuth:userid myID][outRequestBasicAuth:password myPW][outRequestProxy:userid myID]
[outRequestProxy:password myPW][outRequestProxy:port 9061][outRequestProxy:host hostname]]"]')
```

JMSTransport policy and binding properties

Use the -attributes parameter for the setPolicyType and setBinding commands to specify additional configuration information for the JMSTransport policy and policy set binding. Application policy sets can use the JMSTransport policy and binding.

Use the following commands and parameters in the PolicySetManagement group of the AdminTask object to customize your policy set configuration.

- Use the **attributes** parameter for the getPolicyType and getBinding commands to view the properties for your policy and binding configuration. To get an attribute, pass the property name to the getPolicyType or getBinding command.
- Use the **attributes** parameter for the setPolicyType and setBinding commands to add, update, or remove properties from your policy and binding configurations. To add or update an attribute, specify the property name and value. The setPolicyType and setBinding commands update the value if the attribute exists, or adds the attribute and value if the attribute does not exist. To remove an attribute, specify the value as an empty string (""). The **attributes** parameter accepts a properties object.

Note: If a property name or value supplied with the **attributes** parameter is not valid, then the setPolicyType and setBinding commands fail with an exception. The property that is not valid is logged as an error or warning in the `SystemOut.log` file. However, the command exception might not contain the detailed information for the property that caused the exception. When the setPolicyType and setBinding commands fail, examine the `SystemOut.log` file for any error and warning messages that indicate that the input for the **attributes** parameter contains one or multiple properties that are not valid.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global

security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

The following sections explain the policy and binding properties to configure:

- JMSTransport policy properties
- JMSTransport binding properties

JMSTransport policy properties

Use the JMSTransport policy set to configure JMS transport for applications that use the Java Messaging Service (JMS) to exchange request and response messages.

Configure the JMSTransport policy by specifying the following attributes with the **setPolicyType** command:

requestTimeout
Specifies the request timeout value. The request timeout value is the amount of time, in seconds, that the client waits for a response after sending the request to the server. The default value is `300` seconds. If you specify an integer value of zero or less, the system sets the requestTimeout property to the default value of `300` seconds. No maximum value exists for this property.

allowTransactionalAsyncMessaging
Specifies whether a client uses transactions in one-way or asynchronous two-way requests. The default value for this property is `false`. Set the value of this property to `true` to enable transactional messaging. When enabled, the client runtime exchanges SOAP request and response messages with the server over the JMS transport in a transactional manner if the client operates under a transaction.

The client transaction is used to send the SOAP request message to the destination queue or topic, and the server receives the request message only after the client commits the transaction. Similarly, the server receives the request message under the control of a container-managed transaction and sends the reply message, if applicable, back to the client using that same transaction. Then, the client receives the reply message after the server transaction is committed.

The following setPolicyType example command sets values for each JMSTransport binding property:

```
AdminTask.setPolicyType('[-policySet "JMS custom" -policyType JMSTransport
 -attributes "[[requestTimeout 300][allowTransactionalAsynMessaging false]]"]')
```

JMSTransport binding properties

Configure the JMSTransport binding by specifying the following attributes with the **setBinding** command:

outRequestBasicAuth:userid
Specifies the user name or basic authentication of outbound service requests.

outRequestBasicAuth:password
Specifies the password for basic authentication of outbound service requests.

The following **setBinding** example command sets values for each HTTPTransport binding property:

```
AdminTask.setBinding('[-bindingLocation "" -bindingName generalCellWideBind1
 -policyType JMSTransport -attributes "[[outRequestBasicAuth:userid myID] [outRequestBasicAuth:password myPW]]"]')
```

SecureConversation command group for the AdminTask object (Deprecated)

Use this topic as a reference for the commands for the SecureConversation group of the AdminTask object. Use these commands with your administrative scripts to query, update, and remove secure conversation client cache configuration data.

Note: The commands in the SecureConversation command group are deprecated in WebSphere Application Server Version 7.0. Use the commands in the WSSCacheManagement command group to manage WS-Security distributed cache configurations.

Use the following commands in the SecureConversation group to manage your custom and non-custom secure conversation client cache configurations:

- "querySCClientCacheConfiguration command"
- "querySCClientCacheCustomConfiguration command"
- "updateSCClientCacheConfiguration command" on page 567
- "updateSCClientCacheCustomConfiguration command" on page 568
- "deleteSCClientCacheConfigurationCustomProperties command" on page 568

querySCClientCacheConfiguration command

The querySCClientCacheConfiguration command lists all non-custom client cache configuration data for WS-SecureConversation.

Target object

None

Required parameters

None.

Optional parameters

None.

Return value

This command returns a list of all non-custom client cache configuration data.

Batch mode example usage

- Using Jython:

```
print AdminTask.querySCClientCacheConfiguration()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySCClientCacheConfiguration('-interactive')
```

querySCClientCacheCustomConfiguration command

The querySCClientCacheCustomConfiguration command lists all custom client cache configuration data for WS-SecureConversation.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a list of all custom client cache configuration data.

Batch mode example usage

- Using Jython:

```
print AdminTask.querySCClientCacheCustomConfiguration()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySCClientCacheCustomConfiguration('-interactive')
```

updateSCClientCacheConfiguration command

The updateSCClientCacheConfiguration command sets the cache cushion time in minutes and enables or disables distributed cache.

Target object

None.

Required parameters

None.

Optional parameters

-distributedCache
Specifies whether distributed cache is enabled or disabled. If you set the -distributedCache parameter to `true` when you run the updateSCClientCacheConfiguration command, the system enables distributed cache for the WS-Security runtime. (Boolean, optional)

-minutesInCacheAfterTimeout
Specifies the amount of time, in minutes, that the token remains in the cache after it expires. The token is renewable for this amount of time. (Integer, optional)

-renewIntervalBeforeTimeoutMinutes
Specifies the amount of time, in minutes, that a renew request is allowed before the token expires. (Integer, optional)

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:
  ```
  AdminTask.updateSCClientCacheConfiguration('-minutesInCacheAfterTimeout 100 -distributedCache true')
  ```
- Using Jython list:
  ```
  AdminTask.updateSCClientCacheConfiguration(['-minutesInCacheAfterTimeout', '100', '-distributedCache', 'true'])
  ```

Interactive mode example usage
- Using Jython:
  ```
  AdminTask.updateSCClientCacheConfiguration('-interactive')
  ```

updateSCClientCacheCustomConfiguration command

The updateSCClientCacheCustomConfiguration command updates custom properties for the secure conversation client cache configuration.

Target object

None.

Required parameters

None.

Optional parameters

-customProperties
The custom properties for the secure conversation client cache configuration. (Properties, optional)

Return value

This command returns a success or failure message.

Batch mode example usage
- Using Jython string:
  ```
  AdminTask.updateSCClientCacheCustomConfiguration('[-customProperties "[ [property2 value2] [property1 value1] ]"]')
  ```
- Using Jython list:
  ```
  AdminTask.updateSCClientCacheCustomConfiguration(['-customProperties', '[ [property2 value2] [property1 value1] ]'])
  ```

Interactive mode example usage
- Using Jython:
  ```
  AdminTask.updateSCClientCacheCustomConfiguration('-interactive')
  ```

deleteSCClientCacheConfigurationCustomProperties command

The deleteSCClientCacheConfigurationCustomProperties command removes specific properties from a custom secure conversation client cache configuration.

Target object

None.

Required parameters

-propertyNames
The names of the properties to delete. (String, required).

Optional parameters

None.

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:
  `AdminTask.deleteSCClientCacheConfigurationCustomProperties('[-propertyNames [property1,property2]]')`
- Using Jython list:
  `AdminTask.deleteSCClientCacheConfigurationCustomProperties(['-propertyNames', '[property1,property2]'])`

Interactive mode example usage

- Using Jython:
  `AdminTask.deleteSCClientCacheConfigurationCustomProperties('-interactive')`

WSSCacheManagement command group for the AdminTask object

Use this topic as a reference for the commands for the WSSCacheManagement group of the AdminTask object. Use these commands with your administrative scripts to query, update, and remove distributed cache configuration data.

Use the following commands in the WSSCacheManagement group to manage your custom and non-custom distributed cache configurations:

- "deleteWSSDistributedCacheConfigCustomProperties command"
- "queryWSSDistributedCacheConfig command" on page 570
- "queryWSSDistributedCacheCustomConfig command" on page 571
- "updateWSSDistributedCacheConfig command" on page 571
- "updateWSSDistributedCacheCustomConfig command" on page 572

deleteWSSDistributedCacheConfigCustomProperties command

The deleteWSSDistributedCacheConfigCustomProperties command removes WS-Security distributed cache custom properties.

Target object

None

Required parameters

-propertyNames
Specifies the names of the custom properties to delete from the distributed cache configuration. (String[])

Optional parameters

None.

Return value

This command returns a message that indicates the success or failure of the command.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('[-propertyNames [prop1,prop2,prop3]]')
```

- Using Jython list:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties([',-'propertyNames', '[prop1,prop2,prop3]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteWSSDistributedCacheConfigCustomProperties('-interactive')
```

queryWSSDistributedCacheConfig command

The queryWSSDistributedCacheConfig command lists the WS-Security distributed cache configuration non-custom properties.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a properties object that contains the configuration properties and values for the distributed cache configuration. The following table displays the configuration properties that the command returns:

Property	Description
tokenRecovery	Specifies whether token recovery is enabled or disabled. If the tokenRecovery property is set to `true`, the Datasource property specifies the shared data source that is assigned to the distributed cache.
distributedCache	Specifies whether distributed caching is enabled or disabled.
Datasource	Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled.
renewIntervalBeforeTimeoutMinutes	Specifies the amount of time, in minutes, that the client waits before it attempts to renew the token.
synchronousClusterUpdate	Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled.
minutesInCacheAfterTimeout	Specifies the amount of time that the token remains in the cache after the token times out.

Batch mode example usage

- Using Jython:

```
print AdminTask.queryWSSDistributedCacheConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheConfig('-interactive')
```

queryWSSDistributedCacheCustomConfig command

The queryWSSDistributedCacheCustomConfig command lists the WS-Security distributed cache configuration custom properties.

Target object

None.

Required parameters

None.

Optional parameters

None.

Return value

This command returns a properties object that contains the name and value pairs that correspond to each custom property.

Batch mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheCustomConfig()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.queryWSSDistributedCacheCustomConfig('-interactive')
```

updateWSSDistributedCacheConfig command

The updateWSSDistributedCacheConfig command updates the WS-Security distributed cache configuration non-custom properties.

Target object

None.

Required parameters

None.

Optional parameters

-renewIntervalBeforeTimeoutMinutes
Specifies the amount of time, in minutes, that a renew request is allowed before the token expires. (Integer)

-minutesInCacheAfterTimeout
Specifies the amount of time, in minutes, that the token remains in the cache after it expires. The token is renewable for this amount of time. (Integer)

-distributedCache
Specifies whether distributed cache is enabled or disabled. (Boolean)

-synchronousClusterUpdate
Specifies whether the system performs a synchronous update of distributed caches on cluster members. By default, synchronous cluster updating is enabled. Specify `false` to disable synchronous cluster updating. (Boolean)

-tokenRecovery
Specifies whether token recovery is enabled or disabled. If you set the tokenRecovery property to `true`, specify the shared data source to assign to the distributed cache with the -Datasource parameter. (Boolean)

-Datasource
Specifies the name of the shared data source that is assigned to the distributed cache if token recovery is enabled. (String)

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:
  ```
  AdminTask.updateWSSDistributedCacheConfig('[-customProperties "[ [property2 value2] [property1 value1] ]"]')
  ```
- Using Jython list:
  ```
  AdminTask.updateWSSDistributedCacheConfig(['-customProperties', '[ [property2 value2] [property1 value1] ]'])
  ```

Interactive mode example usage

- Using Jython:
  ```
  AdminTask.updateWSSDistributedCacheConfig('-interactive')
  ```

updateWSSDistributedCacheCustomConfig command

The updateWSSDistributedCacheCustomConfig command updates the WS-Security distributed cache configuration custom properties.

Target object

None.

Required parameters

-customProperties
Specifies the name and value of each custom property to add or update in the WS-Security distributed cache configuration. (java.util.Properties)

Optional parameters

None.

Return value

This command returns a success or failure message.

Batch mode example usage

- Using Jython string:
  ```
  AdminTask.updateWSSDistributedCacheCustomConfig('[-customProperties [[property1 value1][property2 value2]]]')
  ```
- Using Jython list:

```
AdminTask.updateWSSDistributedCacheCustomConfig(['-customProperties', '[[property1 value1][property2 value2]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateWSSDistributedCacheCustomConfig('-interactive')
```

PolicySetManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to manage policy set configurations with the wsadmin tool. Use the commands and parameters in the PolicySetManagement group to create, delete, and manage policy set, policy, and policy set attachment configurations.

Before you use the commands in this topic, verify that you are using the most recent version of the wsadmin tool. The policy set management commands that accept a properties object as the value for the **attributes** or **bindingLocation** parameters are not supported on previous versions of the wsadmin tool. For example, the commands do not run on a Version 6.1.0.x node.

Use the following command to manage policy set configurations:

Use the following command to manage policy settings:

Use the following commands to manage policy set attachments:

Use the following commands to manage policy set bindings:

listPolicySets

The listPolicySets command returns a list of all existing policy sets. If administrative security is enabled, each user role can use this command.

Target object

None.

Optional parameters

-policySetType
Specifies the type of policy set. Specify `application` to display application policy sets. Specify `system` to display system policy sets for trust service or WS-MetadataExchange attachments. Specify `system/trust` to display the policy sets for the trust service. Specify `default` to display the default policy sets. The default value for this parameter is `application`. (String, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of all existing policy sets. Each entry in the list is the name of a policy set.

Batch mode example usage

- Using Jython string:

```
AdminTask.listPolicySets('[-policySetType system/trust]')
```

- Using Jython list:

```
AdminTask.listPolicySets(['-policySetType', 'system/trust'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listPolicySets('-interactive')
```

getPolicySet

The getPolicySet command returns general attributes, such as description and default indicator, for the specified policy set. If administrative security is enabled, each user role can use this command.

Target object

None.

Required parameters

-policySet
Specifies the policy set name. For a list of all policy set names, use the listPolicySets command. (String, required)

Optional parameters

-isDefaultPolicySet
Specifies whether to display a default policy set. The default value is `false`. (Boolean, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of attributes for the specified policy set name.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.getPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicySet('-interactive')
```

createPolicySet

The createPolicySet command creates a new policy set. Policies are not created with the policy set. The default indicator is set to `false`.

If administrative security is enabled, you must use the Administrator role to create policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set. (String, required)

Optional parameters

-description
Adds a description for the policy set. (String, required)

-policySetType
Specifies the type of policy set. When the value is `application`, the command creates application policy sets. When the value is `system`, the command creates a policy set that you can use for trust service or WS-MetadataExchange attachments. When the value is `system/trust`, the command creates a policy set for the trust service. The default value for this parameter is `application`. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.createPolicySet('[-policySet myCustomPS -description  [my new custom policy
 set] -policySetType system/trust]')
```

- Using Jython list:

```
AdminTask.createPolicySet(['-policySet', 'myCustomPS', '-description',  '[my new custom
 policy set]', '-policySetType', 'system/trust'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createPolicySet('-interactive')
```

copyPolicySet

The copyPolicySet command creates a copy of an existing policy set. By default, the policy set attachments are transferred to the new policy set.

If administrative security is enabled, you must use the Administrator role to copy policy sets.

Target object

None.

Required parameters

-sourcePolicySet
Specifies the name of the existing policy set to copy. (String, required)

-newPolicySet
Specifies the name of the new policy set you are creating. (String, required)

-newDescription
Specifies a description for the new policy set. (String, required)

Optional parameters

-transferAttachments
If this parameter is set to `true`, all attachments transfer from the source policy set to the new policy set. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.copyPolicySet('[-sourcePolicySet SecureConversation -newPolicySet
 CustomSecureConversation -newDescription [my new copied policy set] -transferAttachments true]')
```

- Using Jython list:

```
AdminTask.copyPolicySet(['-sourcePolicySet', 'SecureConversation', '-newPolicySet',
 'CustomSecureConversation', '-newDescription', '[my new copied policy set]',  '-transferAttachments',
 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.copyPolicySet('-interactive')
```

deletePolicySet

The deletePolicySet command deletes the specified policy set. If attachments exist for the policy set, the command returns a failure message.

If administrative security is enabled, you must use the Administrator role to delete policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to delete. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicySet('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.deletePolicySet(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicySet('-interactive')
```

updatePolicySet

The updatePolicySet command enables you to input an attribute list to update the policy set. You can use this command to update all attributes for the policy set, or a subset of attributes.

If administrative security is enabled, you must use the Administrator role to update policy set configurations.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-attributes
Specifies a properties object that contains the attributes to update for the specified policy set. (Properties, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updatePolicySet('-policySet policySet1 -attributes [[type application]
 [description [my policy set description]]]')
```

- Using Jython list:

```
AdminTask.updatePolicySet(['-policySet', 'policySet1', '-attributes', '[[type
 application] [description [my policy set description]]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updatePolicySet('-interactive')
```

addPolicyType

The addPolicyType command adds a policy with default values for the specified policy set. You must indicate whether to enable or disable the added policy.

If administrative security is enabled, you must use the Administrator role to add policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-policyType
Specifies the name of the policy to add to the policy set. (String, required)

-enabled
If this parameter is set to `true`, new policy is enabled in the policy set. If this parameter is set to `false`, the configuration is contained within the policy set but the configuration does not have an effect on the system. (Boolean, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addPolicyType('[-policySet customPolicySet -policyType WSTransaction
 -enabled true]')
```

- Using Jython list:

```
AdminTask.addPolicyType(['-policySet', 'customPolicySet', '-policyType',
 'WSTransaction', '-enabled', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addPolicyType('-interactive')
```

deletePolicyType

The deletePolicyType command deletes a policy from a policy set.

If administrative security is enabled, you must use the Administrator role to remove policies from your configuration.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to update. (String, required)

-policyType
Specifies the name of the policy to remove from the policy set. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicyType('[-policySet customPolicySet -policyType WSTransaction]')
```

- Using Jython list:

```
AdminTask.deletePolicyType(['-policySet', 'customPolicySet', '-policyType',
 'WSTransaction'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicyType('-interactive')
```

validatePolicySet

The validatePolicySet command validates the policy set configuration.

If administrative security is enabled, you must use the Administrator role to validate policy sets.

Target object

None.

Required parameters

-policySet
Specifies the policy set to update. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.validatePolicySet('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.validatePolicySet(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.validatePolicySet('-interactive')
```

exportPolicySet

The exportPolicySet command exports a policy set as an archive that can be copied onto a client environment.

If administrative security is enabled, you must use the Administrator role to export policy sets.

Target object

None.

Required parameters

-policySet
Specifies the policy set to export. (String, required)

-pathName
Specifies the path name of the archive file to create. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportPolicySet('[-policySet customSecureConversation -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip]')
```

- Using Jython list:

```
AdminTask.exportPolicySet(['-policySet', 'customSecureConversation;, '-pathName', '
 C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportPolicySet('-interactive')
```

importPolicySet

The importPolicySet command imports a policy set from a compressed archive file or from a selection of default policy sets onto the server environment.

If administrative security is enabled, you must use the Administrator role to import policy sets.

Target object

None.

Optional parameters

-importFile
Specifies the path name of the archive file to import. (String, optional)

-defaultPolicySet
Specifies the name of the default policy set to import. (String, optional)

-policySet
Specifies the name to assign to the new policy set. If you do not specify this parameter, the system uses the original name of the policy set. (String, optional)

-verifyPolicySetType
Specifies that the policy set type to import matches a specific type. Specify `system` or `system/trust` to verify that the policy set to import is a type of system policy set, including trust service policy sets. Specify `application` to verify that the policy set is an application policy set. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.importPolicySet('[-importFile C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip]')
```

- Using Jython list:

```
AdminTask.importPolicySet(['-importFile', 'C:/IBM/WebSphere/AppServer/PolicySets/customSC.zip'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importPolicySet('-interactive')
```

listPolicyTypes

The listPolicyTypes command returns a list of the names of the policies configured on your system. The input parameters allow you to list each policy type configured in the system, the policy types configured in a policy set, or the policy types in a binding.

If administrative security is enabled, each administrative role can list policy types.

Target object

None.

Optional parameters

-policySet
Specifies the name of the policy set to query for policies. If the policy set is not specified, the command lists all policies defined in your configuration. (String, optional)

-bindingLocation
Specifies the location of the binding. This value is cell-wide default binding, server-specific default binding, or attachment-specific binding. Specify the bindingLocation parameter as a properties object following these guidelines:

- For cell-wide default binding, use a null or empty properties.
- For server-specific default binding, specify the node and server names in the properties. The property names are `node` and `server`. Server-specific default bindings are deprecated.
- For attachment-specific binding, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For system bindings, set the `systemType` property as `trustService`.
- For WSNClient binding, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`.

(Properties, optional)

-attachmentType
Specifies whether the attachment type is an application binding, client binding, trust service binding, or WS-Notification client binding. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-bindingName
Specifies a specific general binding. If you specify this parameter, the system displays policy types in the specific binding. (String, optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a list of policy types.

Batch mode example usage

- Using Jython string:

```
AdminTask.listPolicyTypes('[-policySet customSecureConversation]')
```

- Using Jython list:

```
AdminTask.listPolicyTypes(['-policySet', 'customSecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listPolicyTypes('[-interactive]')
```

getPolicyType

The getPolicyType command returns the attributes for a specified policy.

If administrative security is enabled, each administrative role can query attributes for policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to query. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

Optional parameters

-attributes
Specifies the specific attributes to display. If this parameter is not used, the command returns all attributes for the specified policy. (String[], optional)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a properties object containing the policy attributes.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicyType('[-policySet customSecureConversation -policyType
 SecureConversation]')
```

- Using Jython list:

```
AdminTask.getPolicyType(['-policySet', 'customSecureConversation', '-policyType', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicyType (['-interactive'])
```

setPolicyType

The setPolicyType command updates the attributes of a specified policy.

Note: The administrative console command assistance provides incorrect Jython syntax for the setPolicyType command. The XPath expression for the response message part protection of the Username WSSecurity policy set contains single quotes (') within each XPath property value, which Jython does not support. To fix the command from the administrative console command assistance, add a backslash character (\) before each single quote to escape the single quote.

If administrative security is enabled, you must use the Administrator role to configure policies.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributes
Specifies the specific attributes to be updated. The properties could include all of the policy attributes or a subset of attributes. (Properties, required)

Optional parameters

-replace
Indicates whether the new attributes provided from the command replace the existing policy attributes. For policies with complex data, you can remove optional parts of the configuration when necessary. Use this parameter to get all attributes, perform edits, and replace the binding configuration with the edited data. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.setPolicyType('[-policySet customSecureConversation -policyType
 SecureConversation -attributes [[type application] [description [my new description]]]]')
```

- Using Jython list:

```
AdminTask.setPolicyType(['-policySet', 'customSecureConversation', '-policyType', 'SecureConversation', '-attributes',
 '[[type application] [description [my new description]]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setPolicyType('-interactive')
```

getPolicyTypeAttribute

The getPolicyTypeAttribute command returns the value for the specified policy attribute.

If administrative security is enabled, each administrative role can query policy type attribute values.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributeName
Specifies the name of the attribute of interest. (String, required)

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Optional parameters

-fromDefaultRepository
Specifies whether to use the default repository. (Boolean, optional)

Return value

The command returns a string that contains the value of the specified attribute.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicyTypeAttribute('[-policySet customSecureConversation  -policyType
 SecureConversation -attributeName type]')
```

- Using Jython list:

```
AdminTask.getPolicyTypeAttribute(['-policySet', 'customSecureConversation',  '-policyType',
 'SecureConversation', '-attributeName', 'type'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getPolicyTypeAttribute('-interactive')
```

setPolicyTypeAttribute

The setPolicyTypeAttribute command sets the value for the specified policy attribute.

If administrative security is enabled, you must use the Administrator role to configure policy attributes.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

-policyType
Specifies the name of the policy of interest. (String, required)

-attributeName
Specifies the name of the attribute of interest. (String, required)

-attributeValue
Specifies the value of the attribute of interest. (String, required)

Return value

If the attribute is successfully added to the policy, the command returns the `true` string value.

Batch mode example usage

- Using Jython string:

```
AdminTask.setPolicyTypeAttribute('[-policySet customPolicySet  -policyType
 WSReliableMessaging -attributeName specLevel  -attributeValue 1.0]')
```

- Using Jython list:

```
AdminTask.setPolicyTypeAttribute(['-policySet', 'customPolicySet',  '-policyType',
 'WSReliableMessaging', '-attributeName', 'specLevel',  '-attributeValue', '1.0'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setPolicyTypeAttribute('-interactive')
```

getPolicySetAttachments

The getPolicySetAttachments command lists the properties for all policy set attachments configured in a specified application.

If administrative security is enabled, each administrative role can query for policy set attachments.

Target object

None.

Optional parameters

-applicationName
Specifies the name of the application to query for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required to query for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-expandResources
Provides expanded information that details the attachment properties for each resource. An asterisk (*

) character returns all Web services. This parameter is valid if the value for the -attachmentType parameter is set to `provider` or `client`. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. (Properties, optional)

Return value

The command returns a list of properties for each attachment in the application, including the policy set name, attachment ID, and resource list. If you specify the expandResources parameter, the command returns the resource, attachmentId, policySet, binding, and directAttachment properties. If a resource is not attached to a policy set, then the system only displays the resource property. The binding property only exists if the attachment contains a custom binding.

Batch mode example usage

- Using Jython string:

```
AdminTask.getPolicySetAttachments('[-attachmentType provider -attachmentProperties "[systemType trustService]"]')
```

- Using Jython list:

```
AdminTask.getPolicySetAttachments(['-attachmentType', 'provider', '-attachmentProperties',
 '[systemType trustService]'])
```

Interactive mode example usage

- Using Jython list:

```
AdminTask.getPolicySetAttachments('-interactive')
```

createPolicySetAttachment

The createPolicySetAttachment command creates a new policy set attachment for an application.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to create policy set attachments. If you have access to a specific resource only, you can create policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can create policy set attachments for application resources only.
Operator	The Operator role cannot create policy set attachments.
Monitor	The Monitor role cannot create policy set attachments.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set to attach. (String, required)

-resources
Specifies the name of the application resources to attach to the policy set. (String[], required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to `true`, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. (Properties, optional)

Return value

The command returns a string with the ID of the new attachment.

Batch mode example usage

- Using Jython string:

```
AdminTask.createPolicySetAttachment('[-policySet policyset1 -resources "WebService:/" -applicationName WebService -attachmentType provider]')
```

- Using Jython list:

```
AdminTask.createPolicySetAttachment(['-policySet', 'policyset1', '-resources', '"WebService:/"', '-applicationName', 'WebService',
 '-attachmentType', 'provider'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createPolicySetAttachment('-interactive')
```

updatePolicySetAttachment

The updatePolicySetAttachment command updates the resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure policy set attachments. If you have access to a specific resource only, you can configure policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to configure policy set attachments. If you have access to a specific resource only, you can configure policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can configure policy set attachments for application resources only.
Operator	The Operator role cannot configure policy set attachments.
Monitor	The Monitor role cannot configure policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to update. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to `true`, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updatePolicySetAttachment('[-attachmentId 123 -resources "WebService:/"]')
```

- Using Jython list:

```
AdminTask.updatePolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updatePolicySetAttachment ('-interactive')
```

addToPolicySetAttachment

The addToPolicySetAttachment command adds additional resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to add resources to policy set attachments. If you have access to a specific resource only, you can add resources to policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to add resources to policy set attachments. If you have access to a specific resource only, you can add resources to policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can add resources to policy set attachments for application resources only.
Operator	The Operator role cannot add resources to policy set attachments.
Monitor	The Monitor role cannot add resources to policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to update. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set

attachments, specify the provider value for the attachmentType parameter and the "[systemType trustService]" value for the -attachmentProperties parameter. For WSNClient attachments, specify the client value for the attachmentType parameter and the bus and WSNService properties with the -attachmentProperties parameter.

-dynamicClient
Set this parameter to true, the system will not recognize the client resources. This option specifies that the client resources are not validated. (Boolean, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as client, and use the -attachmentProperties parameter to specify the bus and WSNService properties. For system policy set attachments, specify the attachmentType parameter as provider, and use the -attachmentProperties parameter to set the systemType property value to trustService. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addToPolicySetAttachment('[-attachmentId 123 -resources
 "WebService:/webapp1.war:{http://www.ibm.com}myService"]')
```

- Using Jython list:

```
AdminTask.addToPolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/webapp1.war:{http://www.ibm.com}myService"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addToPolicySetAttachment('-interactive')
```

removeFromPolicySetAttachment

The removeFromPolicySetAttachment command removes resources that apply to a policy set attachment.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to remove resources from policy set attachments. If you have access to a specific resource only, you can remove resources for which you have access.
Configurator	The Configurator role must have cell-wide access to remove resources from policy set attachments. If you have access to a specific resource only, you can remove the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can remove resources from policy set attachments for application resources only.
Operator	The Operator role cannot remove resources from policy set attachments.
Monitor	The Monitor role cannot remove resources from policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to remove. (String, required)

-resources
Specifies the names of the application resources to attach to the policy set. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.removeFromPolicySetAttachment('[-attachmentId 123 -resources
 "WebService:/webapp1.war:{http://www.ibm.com}myService"]')
```

- Using Jython list:

```
AdminTask.removeFromPolicySetAttachment(['-attachmentId', '123', '-resources',
 '"WebService:/webapp1.war:{http://www.ibm.com}myService"'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeFromPolicySetAttachment('-interactive')
```

deletePolicySetAttachment

The deletePolicySetAttachment command removes a policy set attachment from an application.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can delete policy set attachments for application resources only.
Operator	The Operator role cannot delete policy set attachments.
Monitor	The Monitor role cannot delete policy set attachments.

Target object

None.

Required parameters

-attachmentId
Specifies the name of the attachment to delete. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest for policy set attachments. For application and client attachments, this parameter is required. This parameter is not required for trust service attachments. (String, optional)

-attachmentType
Specifies the type of policy set attachments. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attachmentProperties
Specifies information that is required to identify the location of the attachment. For WSNClient attachments, specify the attachmentType parameter as `client`, and use the -attachmentProperties parameter to specify the `bus` and `WSNService` properties. For system policy set attachments, specify the attachmentType parameter as `provider`, and use the -attachmentProperties parameter to set the `systemType` property value to `trustService`. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deletePolicySetAttachment('[-attachmentId 123]')
```

- Using Jython list:

```
AdminTask.deletePolicySetAttachment(['-attachmentId', '123'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deletePolicySetAttachment('-interactive')
```

listAssetsAttachedToPolicySet

The listAssetsAttachedToPolicySet command lists the applications or WS-Notification service clients to which a specific policy set is attached.

If administrative security is enabled, each administrative role can list applications that are attached to policy sets.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachments. The value for this parameter must be `provider`, `client`, `WSNClient`, `WSMex`, or `all`. The default value is `all`. (String, optional)

Return value

The command returns a list of properties that describe each asset. Each properties object contains the `assetType` property, which specifies the type of asset.

Batch mode example usage

- Using Jython string:

```
AdminTask.listAssetsAttachedToPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.listAssetsAttachedToPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAssetsAttachedToPolicySet('-interactive')
```

listAttachmentsForPolicySet

The listAttachmentsForPolicySet command lists the applications to which a specific policy set is attached.

If administrative security is enabled, each administrative role can query for policy set attachments.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachments. The value for this parameter must be `application`, `client`, or `system/trust`. The default value is `application`. (String, optional)

Return value

The command returns a list of application names.

Batch mode example usage

- Using Jython string:

```
AdminTask.listAttachmentsForPolicySet('[-policySet SecureConversation]')
```

- Using Jython list:

```
AdminTask.listAttachmentsForPolicySet(['-policySet', 'SecureConversation'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listAttachmentsForPolicySet('-interactive')
```

deleteAttachmentsForPolicySet

The deleteAttachmentsForPolicySet command removes all attachments for a specific policy set.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to delete policy set attachments. If you have access to a specific resource only, you can delete policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can delete policy set attachments for application resources only.
Operator	The Operator role cannot delete policy set attachments.
Monitor	The Monitor role cannot delete policy set attachments.

Target object

None.

Required parameters

-policySet
Specifies the name of the policy set from which to remove the attachments. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest. The command only deletes attachments for the application of interest if you specify this parameter. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. You can specify values for the `bus` and `WSNService` properties. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteAttachmentsForPolicySet('[-policySet customSecureConversation
 -applicationName newApp1]')
```

- Using Jython list:

```
AdminTask.deleteAttachmentsForPolicySet(['-policySet', 'customSecureConversation',
 '-applicationName', 'newApp1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteAttachmentsForPolicySet('-interactive')
```

transferAttachmentsForPolicySet

The transferAttachmentsForPolicySet command transfers all attachments from one policy set to another policy set.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to transfer policy set attachments. If you have access to a specific resource only, you can transfer policy set attachments for the resource for which you have access.
Configurator	The Configurator role must have cell-wide access to transfer policy set attachments. If you have access to a specific resource only, you can transfer policy set attachments for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can transfer policy set attachments for application resources only.
Operator	The Operator role cannot transfer policy set attachments.
Monitor	The Monitor role cannot transfer policy set attachments.

Target object

None.

Required parameters

-sourcePolicySet
Specifies the source policy set from which to copy attachments. (String, required)

-destinationPolicySet
Specifies the name of the policy set to which the attachments are copied. (String, required)

Optional parameters

-applicationName
Specifies the name of the application of interest. The command only transfers attachments for the application of interest if you specify this parameter. (String, optional)

-attachmentProperties
Specifies information that is required to identify the location of the attachment. You can specify values for the `bus` and `WSNService` properties. (Properties, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.transferAttachmentsForPolicySet('[-sourcePolicySet SecureConversation
 -destinationPolicySet customSecureConversation -applicationName newApp1]')
```

- Using Jython list:

```
AdminTask.transferAttachmentsForPolicySet(['-sourcePolicySet', 'SecureConversation',
 '-destinationPolicySet', 'customSecureConversation', '-applicationName', 'newApp1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.transferAttachmentsForPolicySet('-interactive')
```

getBinding

The getBinding command returns the binding configuration for a specified policy and scope. You can use the getBinding command to return a list of available custom bindings, which includes bindings that are and are not referenced by attachments.

If administrative security is enabled, each administrative role can query for binding configuration information.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Target object

None.

Required parameters

-policyType
Specifies the policy of interest. (String, required)

-bindingLocation
Specifies the location of the binding. (Properties, required)

Specify the bindingLocation parameter as a properties object following these guidelines:

- For cell-wide general binding or WebSphere Application Server Version 6.1 cell default bindings, specify a null or empty properties.
- For WebSphere Application Server Version 6.1 server-specific default binding, specify the node and server names in the properties. The property names are `node` and `server`. Server-specific default bindings are deprecated.
- For WebSphere Application Server Version 7.0 server default bindings, specify a null or empty properties. Use the bindingName parameter to identify the binding location.
- For attachment-specific, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For WSNClient bindings, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`. If you specify an asterisk character (*) as the attachment ID, then the command returns the list of binding names that corresponds to the attachment type of interest.
- For system/trust bindings, set the `systemType` property as `trustService`.

Optional parameters

-attachmentType
Specifies the type of policy set attachment. Use this parameter to distinguish between types of attachment custom bindings. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attributes
Specifies the names of the attributes to return. If this parameter is not specified, the command returns all attributes. (String[], optional)

-bindingName
Specifies the binding name of interest. Specify this parameter to display a general cell-level binding or a custom attachment binding. (String, optional)

Return value

The command returns a properties object that contains the requested configuration attributes for the policy binding.

Batch mode example usage

- Using Jython string:

  The following example returns a list of application bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType provider
 -bindingLocation [[application application_name] [attachmentId *]]]')
```

  The following example returns a list of client bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType client
 -bindingLocation [[application application_name] [attachmentId *]]]')
```

  The following example returns a list of system bindings:

```
AdminTask.getBinding('[-policyType WSAddressing -attachmentType provider
 -bindingLocation [[systemType trustService] [application application_name] [attachmentId *]]]')
```

- Using Jython list:

```
AdminTask.getBinding(['-policyType', 'WSAddressing', '-bindingLocation', '""'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getBinding('-interactive')
```

setBinding

The setBinding command updates the binding configuration for a specified policy. Use this command to add a server-specific binding, update an attachment to use a custom binding, edit binding attributes, or to remove a binding configuration.

When administrative security is enabled, verify that you use the correct administrative role, as the following table describes:

Administrative role	Authorization
Administrator	The Administrator role must have cell-wide access to configure bindings. If you have access to a specific resource only, you can configure custom bindings for the resource for which you have access. The Administrator role is the only role that can modify binding configurations.
Configurator	The Configurator role must have cell-wide access to assign and unassign bindings. If you have access to a specific resource only, you can assign and unassign bindings for the resource for which you have access.
Deployer	The Deployer role with cell-wide or resource specific access can assign or unassign bindings for application resources only.
Operator	The Operator role cannot configure bindings.
Monitor	The Monitor role cannot configure bindings.

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general binding that are used as the default for an application server or a security domain.

To support a mixed-cell environment, WebSphere Application Server supports Version 7.0 and Version 6.1 bindings. General cell-level bindings are specific to Version 7.0 Application-specific bindings remain at the version that the application requires. When the user creates an application-specific binding, the application server determines the required binding version to use for application.

Target object

None.

Required parameters

-bindingLocation

Specifies the location of the binding. (Properties, required)

Specify the bindingLocation parameter as a properties object following these guidelines:

- For cell-wide general binding or WebSphere Application Server Version 6.1 cell default bindings, specify a null or empty properties.
- For WebSphere Application Server Version 6.1 server-specific default binding, specify the node and server names in the properties. The property names are `node` and `server`. Server-specific default bindings are deprecated.
- For WebSphere Application Server Version 7.0 server default bindings, specify a null or empty properties. Use the bindingName parameter to identify the binding location.
- For attachment-specific, specify the application name and attachment ID in the properties. The property names are `application` and `attachmentId`.
- For WSNClient bindings, specify the bus name, service name, and attachment ID in the properties. The property names are `bus`, `WSNService`, and `attachmentId`. If you specify an asterisk character (*) as the attachment ID, then the command returns the list of binding names that corresponds to the attachment type of interest.
- For system/trust bindings, set the `systemType` property as `trustService`.

-policyType
Specifies the policy of interest. (String, required)

Optional parameters

-attachmentType
Specifies the type of policy set attachment. Use this parameter to distinguish between types of attachment custom bindings. (String, optional)

Note: The `application` and `system/trust` values for the -attachmentType parameter are deprecated. Specify the `provider` value in place of the `application` value. For system policy set attachments, specify the `provider` value for the attachmentType parameter and the `"[systemType trustService]"` value for the -attachmentProperties parameter. For WSNClient attachments, specify the `client` value for the attachmentType parameter and the `bus` and `WSNService` properties with the -attachmentProperties parameter.

-attributes
Specifies the attribute values to update. This parameter can include all binding attributes for the policy or a subset to update. If the **attributes** parameter is not specified, the command only updates the binding location used by the specified attachment. (Properties, optional)

-bindingName
Specifies the name for the binding. Specify this parameter to assign a new name to an attachment binding or cell-level binding. A name is generated if it is not specified. (String, optional)

-domainName
Specifies the domain name for the binding. This parameter is required when using the command to create and scope a binding to a specific domain other than the administrative security domain. The default value is `global`. (String, optional)

-replace
Specifies whether to replace all of the existing binding attributes with the attributes specified in the command. Use this parameter to remove optional parts of the configuration for policies with complex data. The default value is false. (Boolean, optional)

-remove
Specifies whether to remove a server-specific default binding or to remove a custom binding from an attachment. You cannot remove cell-level default binding. The default value is `false`. (Boolean, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.setBinding('[-policyType WSAddressing -bindingLocation [[application myApplication] [attachmentId 123]] -attributes "[preventWLM false]"
 -attachmentType provider]')
```

- Using Jython list:

```
AdminTask.setBinding(['-policyType', 'WSAddressing', '-bindingLocation', '[[application myApplication] [attachmentId 123]]', '-attributes', '[preventWLM
 false]', '-attachmentType', 'provider'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setBinding('-interactive')
```

getDefaultBindings

The getDefaultBindings command displays the provider and client default bindings if the bindings are set. If the command does not return output, then the system default binding is the current default.

If administrative security is enabled, each administrative role can query for default bindings.

Target object

None.

Optional parameters

-bindingLocation
Specifies the location of the binding. Specify the bindingLocation parameter as a properties object with values for the `node` and `server` properties. (Properties, optional)

-domainName
Specifies the domain name for the binding of interest. This parameter is required if the domain of interest is not in the global security domain and you specified the bindingLocation parameter. The bindingLocation and domainName parameters are mutually exclusive. The default value is `global`. (String, optional)

Return value

The command returns a properties object that contains the names of the provider and client default bindings, if the bindings are set.

Batch mode example usage

- Using Jython string:

```
AdminTask.getDefaultBinding('[-bindingLocation [[node myNode] [server myServer]]]')
```

- Using Jython list:

```
AdminTask.getDefaultBinding(['-bindingLocation', '[[node myNode] [server myServer]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getDefaultBinding('-interactive')
```

getRequiredBindingVersion

The getRequiredBindingVersion command displays the version number of the binding for a specific application.

Target object

None.

Optional parameters

-assetProps
Specifies the name of the application of interest. (Properties, optional)

Return value

The command returns the binding version number as a number, such as `7.0.0.0` or `6.1.0.0`.

Batch mode example usage

- Using Jython string:

```
AdminTask.getRequiredBindingVersion('[-assetProps [[application myApplication]]]')
```

- Using Jython list:

```
AdminTask.getRequiredBindingVersion(['-assetProps', '[[application myApplication]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.getRequiredBindingVersion('-interactive')
```

setDefaultBindings

The setDefaultBindings command to set a binding as the default binding.

If administrative security is enabled, you must use the Administrator role with cell-wide access to configure bindings. If you use the Administrator role and do not have cell-wide access, you can only configure bindings on resources for which you have access.

Target object

None.

Required parameters

-defaultBinding
Specifies the names of the default bindings for the provider, client, or both. (Properties, required)

Optional parameters

-bindingLocation
Specifies the location of the binding. Specify the bindingLocation parameter as a properties object with values for the `node` and `server` properties. (Properties, optional)

-domainName
Specifies the domain name for the binding of interest. This parameter is required if the domain of interest is not in the global security domain and you specified the bindingLocation parameter. The bindingLocation and domainName parameters are mutually exclusive. The default value is `global`. (String, optional)

Return value

The command returns a value of `true` if the command successfully sets the default binding.

Batch mode example usage

- Using Jython string:

```
AdminTask.setDefaultBinding('[-bindingName myDefaultBinding -bindingLocation [[node myNode] [server myServer]]]')
```

- Using Jython list:

```
AdminTask.setDefaultBinding(['-bindingName', 'myDefaultBinding', '-bindingLocation', '[[node
 myNode] [server myServer]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setDefaultBinding('-interactive')
```

exportBinding

The exportBinding command export a general, cell-level binding to an archive file. You can copy this file to a client environment or import the archive to a server environment.

If administrative security is enabled, you must use the Administrator role with cell-wide access to export bindings.

Target object

None.

Required parameters

-bindingName
Specifies the name of the binding to assign as the default binding. If you do not specify this parameter, the system specifies the system default as the default binding. (String, required)

-pathName
Specifies the file path for the archive file to create. (String, required)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.exportBinding('[-bindingName myDefaultBinding -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/Bindings/]')
```

- Using Jython list:

```
AdminTask.exportBinding(['-bindingName', 'myDefaultBinding', '-pathName',
 'C:/IBM/WebSphere/AppServer/PolicySets/Bindings/'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.exportBinding('-interactive')
```

importBinding

The importBinding command imports a general, cell-level binding from a compressed archive file to a server environment.

If administrative security is enabled, you must use the Administrator role with cell-wide access to import bindings.

Target object

None.

Required parameters

-pathName
Specifies the file path for the archive file to import. (String, required)

Optional parameters

-bindingName
Specifies the name of the binding to assign as the imported binding. If you do not specify this parameter, the system specifies the binding name in the archive file. (String, optional)

-domainName
Specifies a new name of the domain of the binding to import. If you do not specify this parameter, the command uses the domain specified in the archive file. (String, optional)

-verifyBindingType
Verifies that the type of binding to import matches a specific binding type. Specify `provider` to verify that the binding to import is a provider binding, or specify `client` to verify that it is a client binding. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.importBinding('[-bindingName myDefaultBinding -pathName
 C:/IBM/WebSphere/AppServer/PolicySets/Bindings/myBinding.ear]')
```

- Using Jython list:

```
AdminTask.importBinding(['-bindingName', 'myDefaultBinding', '-pathName',
 'C:/IBM/WebSphere/AppServer/PolicySets/Bindings/myBinding.ear'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.importBinding('-interactive')
```

copyBinding

The copyBinding command creates a new general, cell-level binding from an existing binding.

If administrative security is enabled, you must use the Administrator role with cell-wide access to copy bindings.

Target object

None.

Required parameters

-sourceBinding
Specifies the name of the existing binding that the system uses to create the new binding. (String, required)

-newBinding
Specifies the name of the binding to create. (String, required)

Optional parameters

-newDescription
Specifies the description text for the new binding. (String, optional)

-domainName
Specifies the domain name for the binding. This parameter is only required if you scope the binding to a domain other than the domain of the source binding. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.copyBinding('[-sourceBinding mySourceBinding -newBinding mySourceCopyBinding]')
```

- Using Jython list:

```
AdminTask.copyBinding(['-sourceBinding', 'mySourceBinding', '-newBinding',
 'mySourceCopyBinding'])
```

Interactive mode example usage

- Using Jython list:

```
AdminTask.copyBinding('-interactive')
```

upgradeBindings

The upgradeBindings command upgrades application bindings for a specific asset to the latest version.

If administrative security is enabled, you must use the Administrator role with cell-wide access to import bindings.

Target object

None.

Required parameters

-assetProps
Specifies the name of the asset of interest. Specify the name of the application as the value for the **application** property. (Properties, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.upgradeBindings('[-assetProps [[application myApplication]]]')
```

- Using Jython list:

```
AdminTask.upgradeBindings(['-assetProps', '[[application myApplication]]'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.upgradeBindings('-interactive')
```

WS-Policy commands for the AdminTask object

You can use the Jython or Jacl scripting languages to manage WS-Policy settings for Web service resources with the wsadmin tool. You can view or manage how a service provider shares its policies, and how a service client obtains and applies the policies of a service provider.

To run these commands, use the AdminTask object of the wsadmin scripting client. Each command acts on multiple objects in one operation. The commands are provided to allow you to make the most commonly-required types of update in a consistent manner, where modifying the underlying objects directly would be error-prone.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

These commands are valid only when they are used with WebSphere Application Server Version 7 and later application servers. Do not use them with earlier versions.

The commands to manage WS-Policy settings for Web service resources are part of the PolicySetManagement command group for the AdminTask object.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using these commands, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

The commands that are listed as subtopics are available to manage WS-Policy settings in the PolicySetManagement group of the AdminTask object.

The following commands are available to manage WS-Policy settings in the PolicySetManagement group of the AdminTask object:

- getProviderPolicySharingInfo command
- setProviderPolicySharingInfo command
- getClientDynamicPolicyControl command
- setClientDynamicPolicyControl command

getProviderPolicySharingInfo command

Use the getProviderPolicySharingInfo command to find out whether an application or service that is a Web service provider can share its policy configuration, and list the properties that apply to sharing that configuration.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the getProviderPolicySharingInfo command to find out how a Web services application, or a service in a Web services application, shares its policy configuration with clients, service registries, or services that support the WS-Policy specification. The policy configuration is shared in WS-PolicyAttachments format.

The command returns properties that show whether the policy configuration of the resource can be shared with clients through a WS-MetadataExchange request or through Web Service Description Language (WSDL) that is obtained by a ?WSDL HTTP Get request.

Target object

None.

Required parameters

-applicationName
The name of the application for which you want to find out how it shares its policy configuration. The application must be a service provider. (String)

Optional parameters

-resource
The name of the resource for which you want to find out how it shares its policy configuration. If you specify this parameter, only the properties for that resource are returned. To retrieve information for the application, specify `WebService:/`. Alternatively, you can specify a service, endpoint or operation. However, policy sets are attached only at the application or service level, so the properties returned for an endpoint or operation are the settings that are inherited from the service. (String)

Return value

Returns a list of properties that include the resource name and that show whether the policy configuration of the resource can be shared. The following properties can be returned:

wsMexPolicySetName
The name of the policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the sharePolicyMethods property is wsMex and a policy set to provide message-level security was specified.

wsMexPolicySetBinding
The name of the binding that is applied when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the sharePolicyMethods property is wsMex and a binding to provide message-level security was specified.

resource
The resource that you specified.

directSetting
How the properties apply to the resource. Valid values for this property are:

true
The properties apply directly to the resource.

false
The properties are inherited from the parent application or service.

sharePolicyMethods
How the policy configuration of the resource can be shared. Valid values for this property are:

httpGet
The resource shares its policy configuration through an HTTP Get request.

wsMex
The resource shares its policy configuration through a WS-MetadataExchange request.

Example

The following command displays the policy sharing configuration properties for the EchoService service in the WSSampleServices application. The provider is configured to share its policy through an HTTP Get request, and a WS-MetadataExchange request with message-level security. Message-level security for the WS-MetadataExchange request is provided using the SystemWSSecurityDefault policy set and the "Provider sample" general binding.

```
AdminTask.getProviderPolicySharingInfo(['-applicationName', 'WSSampleServices',
'-resource', 'WebService:/SampleServicesSei.war:{http://example_path/}EchoService'])
.
.
[ [wsMexPolicySetName SystemWSSecurityDefault] [wsMexPolicySetBinding [Provider sample]]
[resource WebService:/SampleServicesSei.war:{http://example_path/}EchoService/]
[directSetting true] [sharePolicyMethods [httpGet wsMex]] ]
```

setProviderPolicySharingInfo command

Use the setProviderPolicySharingInfo command to set how an application or service that is a Web service provider can share its policy configuration with other clients, service registries, or services that support the WS-Policy specification. You can set or remove this information about how a provider policy is shared.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic "Configure Qshell to run WebSphere Application Server scripts".

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the setProviderPolicySharingInfo command to set how an application, or a service in an application, shares its policy configuration with clients, service registries, or services that support the WS-Policy specification. The policy configuration is shared in WS-PolicyAttachments format.

The policy configuration of the resource can be shared with clients through a WS-MetadataExchange request, through Web Service Description Language (WSDL) exported by a `?WSDL` HTTP Get request, or through both methods.

Target object

None.

Required parameters

-applicationName
The name of the application for which you want to set how the provider policy is shared. (String)

-resource
The name of the resource for which you want to set how the provider policy is shared. For all resources in an application, specify `WebService:/`. For a service in an application, specify `WebService:/`*`module`*`:{`*`namespace`*`}`*`service_name`*. Endpoints or operations inherit the settings of the parent application or service. (String)

Optional parameters

-sharePolicyMethods
Specifies how the policy configuration of the resource can be shared. (String array)

Enter either or both of the following values:

`httpGet`
The resource can share its policy configuration through WSDL that is obtained by a `?WSDL` HTTP Get request.

`wsMex` The resource can share its policy configuration through a WS-MetadataExchange request.

-wsMexProperties
Specifies that message-level security is required for WS-MetadataExchange requests and specifies the settings that provide the message-level security. (Properties)

Enter the following values, following each value with the setting that you require for that value:

`wsMexPolicySetName`
The name of the system policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. Specify a system policy set that contains only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault.

`wsMexPolicySetBinding`
The name of the general binding for the policy set attachment when the resource shares its policy configuration through a WS-MetadataExchange request. Specify a general binding that is scoped to the global domain, or scoped to the security domain of this service. If you do not specify this property, the default binding is used.

This parameter is valid only when you specify `wsMex` for the **sharePolicyMethods** parameter.

-remove
Specifies whether the information about how the provider policy is shared is removed from the resource. (Boolean)

This parameter takes the following values:

true The information about how the provider policy is shared is removed from the resource.

false This value is the default. The information about how the provider policy is shared is not removed from the resource.

Examples

The following example removes the information about how the provider policy is shared from the WSSampleServices application:

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/ -remove true]')
```

The following example enables policy sharing, using WSDL exported by a `?WSDL` HTTP Get request, for the EchoService service in the WSSampleServices application:

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/WSSampleServicesSei.war:{http://example_path/}EchoService
-sharePolicyMethods [httpGet ]]')
```

The following example enables policy sharing, using a WS-MetadataExchange request with message-level security, for the WSSampleServices application. Message level security is provided using the MexPS policy set and the “Client sample” general binding.

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WSSampleServices
-resource WebService:/ -sharePolicyMethods [wsMex ]
-wsMexProperties [ [wsMexPolicySetName [SystemWSSecurityDefault]]
[wsMexPolicySetBinding [Provider sample]] ]]')
```

getClientDynamicPolicyControl command

Use the getClientDynamicPolicyControl command to find out whether an application that is a Web service client obtains the policy configuration of a Web service provider, and to list the properties that apply to obtaining that configuration.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the getClientDynamicPolicyControl command to find out how an application, or a service in an application, obtains the policy configuration of a service provider. The client can obtain the policy configuration of the provider through a Web Services Metadata Exchange (WS-MetadataExchange) request or through an HTTP Get request.

Target object

None.

Required parameters

-applicationName
The name of the application for which you want to find out how it obtains the policy configuration of a service provider. The application must be a service client. (String)

Optional parameters

-resource
The name of the resource for which you want to find out how it obtains the policy configuration of a service provider. If you specify this parameter, only the properties for that resource are returned. To retrieve information for the application, specify `WebService:/`. Alternatively, you can specify a service, endpoint, or operation. However, policy sets are attached only at the application or service level, so the properties returned for an endpoint or operation are the settings that are inherited from the service. (String)

Return value

Returns a list of properties that include the resource name and that show how it obtains the policy configuration of a service provider. The following properties can be returned:

`httpGetTargetURI`
The target URL of the HTTP Get request. This property is returned if the value of the acquireProviderPolicyMethod property is httpGet.

`wsMexPolicySetName`
The name of the policy set that specifies message-level security when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the acquireProviderPolicyMethod property is wsMex and a policy set to provide message-level security was specified.

`wsMexPolicySetBinding`
The name of the binding that is used when the resource shares its policy configuration through a WS-MetadataExchange request. This property is returned if the value of the acquireProviderPolicyMethod property is wsMex and a binding to provide message-level security was specified.

`acquireProviderPolicyMethod`
How the policy configuration of the provider can be obtained. Valid values for this property are:

`wsMex`
The resource can obtain the policy configuration of a service provider through a WS-MetadataExchange request.

`httpGet`
The resource can obtain the policy configuration of a service provider through an HTTP Get request.

resource
The resource that you specified.

directSetting
How the properties apply to the resource. Valid values for this property are:

true
The properties apply directly to the resource.

false
The properties are inherited from the parent application or service.

Examples

The following example displays the properties that control how the EchoService service of the WSPolicyClient application obtains the policy configuration of a service provider. The client is configured to retrieve the provider policy through a WS-MetadataExchange request with message-level security, using the SystemWSSecurityDefault policy set and the “Client sample” general binding.

```
AdminTask.getClientDynamicPolicyControl(['-applicationName', 'WSPolicyClient',
'-resource', 'WebService:/WSPClient.war:{http://example_path/}EchoService'])
.
.
[ [wsMexPolicySetName SystemWSSecurityDefault] [wsMexPolicySetBinding [Client sample]]
[acquireProviderPolicyMethod [wsMex]]
[resource WebService:/WSPClient.war:{http://example_path/}EchoService/]
[directSetting true] ]
```

The following example displays the properties that control how the EchoService service of the WSPolicyClient application obtains the policy configuration of a service provider when the client is configured to retrieve the provider policy through an HTTP Get request.

```
AdminTask.getClientDynamicPolicyControl(['-applicationName', 'WSPolicyClient',
'-resource', 'WebService:/WSPClient.war:{http://example_path/}EchoService'])
.
.
[ [httpGetTargetURI http://example_path/EchoService?wsdl]
[acquireProviderPolicyMethod [httpGet]]
[resource WebService:/WSPClient.war:{http://example_path/}EchoService/]
[directSetting true] ]
```

setClientDynamicPolicyControl command

Use the setClientDynamicPolicyControl command to set how an application that is a Web services client obtains the policy configuration of a Web services provider. You can set, refresh, or remove this information about how a provider policy is obtained.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available policy set management administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('PolicySetManagement')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the setClientDynamicPolicyControl command to set how a client obtains the policy configuration of a service provider.

The client can obtain the policy configuration of the provider through a Web Services Metadata Exchange (WS-MetadataExchange) request or through an HTTP Get request. The service provider must publish its policy in WS-PolicyAttachment format in its Web Service Description Language (WSDL) and the client must be able to support those provider policies.

At run time, the client uses the information to establish a policy configuration that is acceptable to both the client and the service provider.

Target object

An application or service that is a Web services client.

Required parameters

-applicationName
The name of the application for which you want to obtain the policy configuration of the provider. (String)

-resource
The name of the resource for which you want to obtain the policy configuration of the provider. For all resources in an application, specify `WebService:/`. For a service in an application, specify `WebService:/`*`module`*`:{`*`namespace`*`}`*`service_name`*. Endpoints or operations inherit the settings of the parent application or service. (String)

Optional parameters

-acquireProviderPolicyMethod
Specifies how the policy configuration of the provider can be obtained. (String)

Enter one of the following values:

`httpGet`
Obtain the policy configuration of the provider using an HTTP Get request. By default, the HTTP Get request is targeted at the URL for each service endpoint followed by `?WSDL`. If you specify this value for a service, you can use the **httpGetProperties** parameter to change the target of the request.

`wsMex` Obtain the policy configuration of the provider using a WS-MetadataExchange request.

-wsMexProperties
Specifies that message-level security is required for WS-MetadataExchange requests and specifies the settings that provide the message-level security. (Properties)

Enter the following values, following each value with the setting that you require for that value:

`wsMexPolicySetName`
The name of the system policy set that specifies message-level security when the policy configuration of the provider is obtained through a WS-MetadataExchange request. Specify a system policy set that contains only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault.

wsMexPolicySetBinding
The name of the general binding for the policy set attachment when the resource shares its policy configuration through a WS-MetadataExchange request. Specify a general binding that is scoped to the global domain, or scoped to the security domain of this service. If you do not specify this property, the default binding is used.

This parameter is valid only when you specify `wsMex` for the **acquireProviderPolicyMethod** parameter.

-httpGetProperties
Specifies the target for an HTTP Get request for a service if you do not want to use the default. (Properties)

Enter the following value, followed by the setting that you require for that value:

httpGetTargetURI
The URL for the service endpoint followed by `?WSDL`.

This parameter is valid only when you specify `httpGet` for the **acquireProviderPolicyMethod** parameter and the resource is a service. Do not use this parameter if the resource is an application.

-remove
Specifies whether to remove the information about how the client obtains the policy configuration of the provider. (Boolean)

This parameter takes the following values:

true Information about how the client obtains the policy configuration of the provider is removed.

false This value is the default. Information about how the client obtains the policy configuration of the provider is not removed.

Examples

The following example removes the information about how the client obtains the policy configuration of the provider from the EchoService service of the WSPolicyClient client application.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-remove true]')
```

The following example configures the EchoService service of the WSPolicyClient client application to obtain the policy configuration of the provider using an HTTP Get request.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-acquireProviderPolicyMethod [httpGet ]
-httpGetProperties [httpGetTargetURI http://example_path?WSDL]]')
```

The following example configures the EchoService service of the WSPolicyClient client application to obtain the the policy configuration of the provider through a WS-MetadataExchange request with message-level security, using the SystemWSSecurityDefault policy set and the “Client sample” general binding.

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WSPolicyClient
-resource WebService:/WSPolicyClient.war:{http://example_path/}EchoService
-acquireProviderPolicyMethod [wsMex ]
-wsMexProperties [ [wsMexPolicySetName [SystemWSSecurityDefault]]
[wsMexPolicySetBinding [Client sample]] ]]')
```

Configuring secure sessions between clients and services using the wsadmin tool

Use the wsadmin tool, which supports the Jython and Jacl scripting language, to edit trust service configurations. Use the STSManagement command group for the AdminTask object to specify details related to secure sessions between clients and target services.

About this task

The trust service uses the secure messaging mechanisms of the Web Services Trust (WS-Trust) specification to define additional extensions for issuing, exchanging, and validating security tokens. Use the STSManagement command group for the AdminTask object to configure the trust service using the wsadmin tool. Complete any of the following tasks using the STSManagement commands:

- Manage token provider configurations.

  Use the wsadmin tool to manage token providers. Customize token providers by defining properties such as token type schema URI, handler factory, cache cushion time, class name, and token timeout. You can also allow or restrict the use of post-dated tokens, distributed cache, and renewable tokens after timeout.
- Query existing token provider configurations.

  Use the wsadmin tool to query the existing trust service token provider configuration.
- Manage endpoint token assignments.

  Use the wsadmin tool to assign, unassign, and modify endpoint token assignments.
- Refresh your configuration changes.

  Use the wsadmin tool to force the trust service to reload the token provider configuration during run time. Complete this action to use new configuration changes before you restart the application server.

What to do next

Use the information center topics for managing token providers using the STSManagement group of commands and the AdminTask object.

Querying the trust service using scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to query the trust service for existing configuration settings. Use the commands in this topic to view current trust service configurations before adding, removing, or editing token provider and endpoint configurations.

About this task

Query your current token provider configurations or endpoint configurations using the STSManagement group of commands. Use the following Jython syntax command examples when writing automation scripts to retrieve configuration attributes and set the output to a variable. Pass the newly set variable to administrative commands in the STSManagement group to automate the editing of token provider and endpoint configurations.

- Use the following command examples to query the trust service for token provider configurations.
  - Determine the local name of the default token provider and set it to the myDefaultTokenType variable.

    The following command sets the *myDefaultTokenType* variable to the local name string for the default token provider:

    ```
    myDefaultTokenType = AdminTask.querySTSDefaultTokenType()
    print myDefaultTokenType
    ```
  - List the local names of each configured token provider.

    The following command sets the *myTokenTypes* variable to an array containing the local names of the configured token providers:

    ```
    myTokenTypes = AdminTask.listSTSConfiguredTokenTypes()
    print myTokenTypes
    ```
  - Display the non-custom properties for the default token provider.

    The following command returns a java.util.Properties instance that contains the values for each non-custom property for the default token provider stored in the *myDefaultTokenType* variable.

    ```
    AdminTask.querySTSTokenTypeConfigurationDefaultProperties(myDefaultTokenType)
    ```

To use this command to query a specific token provider, use the following Jython syntax:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('"Security Context Token"')
```

– Display a properties object containing all custom properties for a token provider configuration.

The following command returns a java.util.Properties instance that contains the values for each of the custom properties for the token provider stored in the *myDefaultTokenType* variable.

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties(myDefaultTokenType)
```

To use this command to query a specific token provider, use the following Jython syntax:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('"Security Context Token"')
```

- Use the following command examples to query endpoint target configurations and security constraints for endpoint targets.
  – Display each uniform resource identifier (URI) for each assigned endpoint.

The following command sets the *allMyURIs* variable to an array containing the URIs for each assigned endpoint:

```
allMyURIs = AdminTask.listSTSAssignedEndpoints()
print allMyURIs
```

– Display the token provider that is assigned to a specific endpoint URI.

The following command sets the *myTokenType* variable to the name of the token provider that is assigned to the `http://myserver.mysom.com:9080/Example` endpoint URI:

```
myTokenType = AdminTask.querySTSEndpointTokenType('http://myserver.mysom.com:9080/Example')
print myTokenType
```

What to do next

Use the wsadmin tool to manage and edit token provider and endpoint configurations.

Related tasks

“Configuring secure sessions between clients and services using the wsadmin tool” on page 613
Use the wsadmin tool, which supports the Jython and Jacl scripting language, to edit trust service configurations. Use the STSManagement command group for the AdminTask object to specify details related to secure sessions between clients and target services.

“Managing existing token providers with scripting”
You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the trust service. Use this topic to modify token provider configuration data, and to add custom properties.

“Associating token providers with endpoint services (targets) using scripting” on page 620
You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the association of endpoints and tokens. Use this topic to query, assign, and unassign the association of a token provider with an endpoint Uniform Resource Identifier (URI).

Related reference

“STSManagement command group for the AdminTask object” on page 622
You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the STSManagement group can be used to manage and query trust service token provider configurations and endpoint configurations.

Related information

Trust service targets collection
Use this page to view a list of targets, which are application server service endpoints. You can manage tokens by specifying which token is to be issued when access to a specific endpoint is requested.

Trust service token providers collection
Use this page to view information about or manage token providers for the trust service.

Managing existing token providers with scripting

You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the trust service. Use this topic to modify token provider configuration data, and to add custom properties.

Before you begin

You must have an existing token provider configured in the trust service.

About this task

Use the commands in the STSManagement group of the AdminTask object to modify existing configuration data. This topics includes examples for modifying existing non-custom configuration data.

Modify existing configuration data.
Use the **updateSTSTokenTypeConfiguration** command to update existing properties for a specific token provider configuration. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.

1. Determine the token provider configuration to edit.

   Enter the following command to view the list of names of the configured token providers:

   ```
   AdminTask.listSTSConfiguredTokenTypes()
   ```

2. Review the current configuration data for the token provider configuration to edit.

   Enter the following command to view a Properties object containing all non-custom configuration data for the Security Context Token token provider:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('Security Context Token')
```

3. Update the token provider configuration with new configuration data.

   Determine which parameters to update in your configuration, using the following table as a reference:

Parameter	Data type
LocalName Specifies the unique token provider name as the target object of the command.	String, required
-HandlerFactory Specifies the configuration class name, including package information.	String, required
-URI Specifies the unique token type schema URI.	String, required
-lifetimeMinutes Specifies the amount of time, in minutes, that the token is valid.	Integer, optional Default: 120 (minutes) Minimum: 10 (minutes)
-renewalWindowMinutes Specifies the amount of time after the token expires during which the token can be renewed.	Integer, optional Default: 120 (minutes) Minimum: 10 (minutes)
-postdatable Set to `true` to specify that tokens of the token provider are valid at a later time. Tokens can be created with or without a future start time.	Boolean, optional Default: false
-distributedCache (deprecated) Set to `true` to enable distributed cache. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.	Boolean, optional Default: false
-renewableAfterExpiration Set to `true` to specify that tokens of the token provider are renewable after expiration.	Boolean, optional Default: false

Parameter	Data type
-tokenCacheFactory (deprecated) Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter.	String, optional Default: `com.ibm.ws.wssecurity.platform.websphere.trust .server.sts.ext.cache.STSTokenCacheFactoryImpl`

Use the **updateSTSTokenTypeConfiguration** command to update the configuration data for the `Security Context Token` token provider. The following example changes the time that the token is valid from 60 minutes to 100 minutes, disables token renewal after expiration, and enables distributed caching:

```
AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-lifetimeMinutes 100 -renewableAfterExpiration false -distributedCache true]')
```

The command returns a message indicating the success or failure of the operation.

4. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

5. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the application server:

   ```
   AdminTask.refreshSTS()
   ```

Related tasks

“Configuring secure sessions between clients and services using the wsadmin tool” on page 613
Use the wsadmin tool, which supports the Jython and Jacl scripting language, to edit trust service configurations. Use the STSManagement command group for the AdminTask object to specify details related to secure sessions between clients and target services.

“Querying the trust service using scripting” on page 614
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to query the trust service for existing configuration settings. Use the commands in this topic to view current trust service configurations before adding, removing, or editing token provider and endpoint configurations.

“Adding and removing token provider custom properties using scripting”
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to administer the trust service. Use this topic to set internal system configuration properties for your token provider configuration by adding or removing custom properties.

“Associating token providers with endpoint services (targets) using scripting” on page 620
You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the association of endpoints and tokens. Use this topic to query, assign, and unassign the association of a token provider with an endpoint Uniform Resource Identifier (URI).

Related reference

Trust service token provider settings
Use this page to modify information for an existing token provider.

“STSManagement command group for the AdminTask object” on page 622
You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the STSManagement group can be used to manage and query trust service token provider configurations and endpoint configurations.

Adding and removing token provider custom properties using scripting

Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to administer the trust service. Use this topic to set internal system configuration properties for your token provider configuration by adding or removing custom properties.

Before you begin

You must have an existing token provider configured for the trust service.

About this task

Use custom properties to set internal system configuration properties and specify these properties using the customProperties parameter. Custom properties are arbitrary name and value pairs of data, where the name can be a property key or a class implementation, and where the value might be a string or Boolean value. Use this topic and the commands in the STSManagement group for the AdminTask object to add or remove custom properties from your configuration with the Jython scripting language.

- Add new custom properties to a specific token provider configuration.

  Use the **updateSTSTokenTypeConfiguration** command to add or update custom properties to your token provider configuration. Do not use the **updateSTSTokenTypeConfiguration** command to remove custom properties. If you specify the -distributedCache parameter, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for the -distributedCache parameter for custom tokens.

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Determine the token provider configuration to edit.

     Enter the following command to view a list of the names for each configured token provider:

     ```
     AdminTask.listSTSConfiguredTokenTypes()
     ```
  3. Review the configured custom properties for the token provider of interest.

     Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

     ```
     AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
     ```
  4. Add custom properties to the token provider configuration.

     Use the **updateSTSTokenTypeConfiguration** command to add the configuration data for the `Security Context Token` token provider. Use the following example to add the `com.ibm.ws.security.webChallengeIfCustomSubjectNotFound` custom property with a value of `false` and the `com.ibm.ws.security.defaultLoginConfig` custom property with a value of `system.DEFAULT` to the configuration:

     ```
     AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-customProperties
     [[com.ibm.ws.security.webChallengeIfCustomSubjectNotFound false]
     [com.ibm.ws.security.defaultLoginConfig system.DEFAULT]] ]')
     ```

     The command returns a message indicating the success or failure of the operation.
  5. Save your configuration changes.

     Use the following command to save your changes:

     ```
     AdminConfig.save()
     ```
  6. Reload the modified configuration changes.

     Use the following command to force the trust service to reload your modified configuration without restarting the application server.

     ```
     AdminTask.refreshSTS()
     ```
- Edit custom properties for a specific token provider configuration.
  1. View configured custom properties for the token provider of interest.

     Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

     ```
     AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
     ```
  2. Modify the configuration data for the token provider of interest.

     Use the **updateSTSTokenTypeConfiguration** command to modify the existing configuration data for the *Security Context Token* token provider. This example specifies that the *Security Context*

Token token provider configuration includes the com.ibm.ws.security.webChallengeIfCustomSubjectNotFound custom property with a value of false and the com.ibm.ws.security.defaultLoginConfig custom property with a value of system.DEFAULT. Use the following command to change the value of the com.ibm.ws.security.defaultLoginConfig custom property from system.DEFAULT to system.CUSTOM, and does not change any other configured custom properties:

```
AdminTask.updateSTSTokenTypeConfiguration('Security Context Token', '[-customProperties
[[com.ibm.ws.security.defaultLoginConfig system.CUSTOM]]]')
```

The command returns a message indicating the success or failure of the operation.

3. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

4. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the application server:

   ```
   AdminTask.refreshSTS()
   ```

- Remove custom properties from token provider configurations.

1. View configured custom properties for the token provider of interest.

   Enter the following command to view a properties object containing custom configuration data for the *Security Context Token* token provider:

   ```
   AdminTask.querySTSTokenTypeConfigurationCustomProperties('Security Context Token')
   ```

2. Delete the custom property from the token provider configuration.

   Use the **deleteSTSTokenTypeConfigurationCustomProperties** command to delete custom properties from your configuration. Specify the names of the custom properties to remove using the propertyNames parameter. If the specified name does not exist in the configuration, no configuration changes are made. The following command removes the com.ibm.ws.security.webChallengeIfCustomSubjectNotFound and com.ibm.ws.security.defaultLoginConfig custom properties from the *Security Context Token* token provider configuration:

   ```
   AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('Security Context Token',
   '[-propertyNames com.ibm.ws.security.webChallengeIfCustomSubjectNotFound
   com.ibm.ws.security.defaultLoginConfig]')
   ```

   The command returns a message indicating the success or failure of the operation.

3. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

4. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the service:

   ```
   AdminTask.refreshSTS()
   ```

Related tasks

“Configuring secure sessions between clients and services using the wsadmin tool” on page 613
Use the wsadmin tool, which supports the Jython and Jacl scripting language, to edit trust service configurations. Use the STSManagement command group for the AdminTask object to specify details related to secure sessions between clients and target services.

“Querying the trust service using scripting” on page 614
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to query the trust service for existing configuration settings. Use the commands in this topic to view current trust service configurations before adding, removing, or editing token provider and endpoint configurations.

“Managing existing token providers with scripting” on page 615
You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the trust service. Use this topic to modify token provider configuration data, and to add custom properties.

“Associating token providers with endpoint services (targets) using scripting”
You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the association of endpoints and tokens. Use this topic to query, assign, and unassign the association of a token provider with an endpoint Uniform Resource Identifier (URI).

Related reference

Trust service token custom properties
WebSphere Application Server trust service provides several custom properties by default to define the default security context token (SCT).

Trust service token provider settings
Use this page to modify information for an existing token provider.

“STSManagement command group for the AdminTask object” on page 622
You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the STSManagement group can be used to manage and query trust service token provider configurations and endpoint configurations.

Associating token providers with endpoint services (targets) using scripting

You can use the wsadmin tool, which supports the Jython and Jacl scripting languages, to manage the association of endpoints and tokens. Use this topic to query, assign, and unassign the association of a token provider with an endpoint Uniform Resource Identifier (URI).

Before you begin

Before you can assign and manage endpoint configurations, at least one token provider configuration and a Web service must exist.

About this task

Use the STSManagement group of commands to specify a custom service endpoint Uniform Resource Identifier (URI) and to assign and unassign the association of trust service token providers with endpoint configurations. Complete the steps in this topic to query the trust service for the existing endpoint configuration, associate the default token with an endpoint, and unassociate a token from an endpoint. You can perform these steps in any order.

- Associate a token with a specific endpoint.
  1. View a list of all endpoint URIs that are currently associated with a token provider.

     Before invoking changes on your endpoint configurations, use the following listSTSAssignedEndpoints command to examine your current settings:

     ```
     AdminTask.listSTSAssignedEndpoints()
     ```

If the endpoint of interest is currently associated with a token, do not use the assignSTSEndpointTokenType command. To update the token that is associated with the endpoint, use the updateSTSEndpointTokenType command in the next step.

2. Associate a token with an endpoint.

   Use the assignSTSEndpointTokenType command to specify the token to issue for access to a specific endpoint. You do not need to specify the name of the token provider to assign if the token provider is set as the default configuration. For example, the following command assigns the `Security Context Token` default token to the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

   ```
   AdminTask.assignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog')
   ```

   If `Security Context Token` is not the default token provider, use the following command:

   ```
   AdminTask.assignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog',
   '-LocalName Security Context Token')
   ```

   The command returns a message indicating the success of the operation.

3. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

4. Reload the modified configuration changes.

   Use the following command to force the trust service to reload your modified configuration without restarting the application server:

   ```
   AdminTask.refreshSTS()
   ```

- Disassociate a token from an endpoint.

1. Examine the current endpoint configuration.

   Use the listSTSAssignedEndpoints to view a list of each endpoint URI with assigned token providers, as the following example describes:

   ```
   AdminTask.listSTSAssignedEndpoints()
   ```

   The following sample output is displayed:

   ```
   'http://www.mycompany.com:8080/Ecommerce/Catalog'
   ```

2. Choose the endpoint to edit.

   Use the querySTSEndpointTokenType to return the token provider associated with the endpoint of interest. Enter the following command to view the token provider associated with the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

   ```
   AdminTask.querySTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog')
   ```

   The following sample output is displayed:

   ```
   'Security Context Token'
   ```

3. Disassociate the token type from the endpoint.

   Use the **unassignSTSEndpointTokenType** command to disassociate the token provider and endpoint configuration. The following command removes the `Security Context Token` token provider that is associated with the `http://www.mycompany.com:8080/Ecommerce/Catalog` endpoint URI:

   ```
   AdminTask.unassignSTSEndpointTokenType('http://www.mycompany.com:8080/Ecommerce/Catalog',
   '-LocalName Security Context Token')
   ```

   The command returns a message indicating the success of the operation.

4. Save your configuration changes.

   Use the following command to save your changes:

   ```
   AdminConfig.save()
   ```

5. Reload the modified configuration changes.

Use the following command to force the trust service to reload your modified configuration without restarting the service:

```
AdminTask.refreshSTS()
```

STSManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure security with the wsadmin tool. The commands and parameters in the STSManagement group can be used to manage and query trust service token provider configurations and endpoint configurations.

The STSManagement command group contains commands that allow you to configure existing token providers, assign token providers to endpoints, and modify general trust service configuration data. The commands in this group that perform configuration changes require that you execute the save command to commit the changes. No configuration changes are made if an exception is created when executing a command.

Use the following commands to modify and query token provider configurations:

- "createSTSTokenTypeConfiguration "
- "deleteSTSTokenTypeConfigurationCustomProperties " on page 623
- "listSTSConfiguredTokenTypes " on page 624
- "querySTSDefaultTokenType " on page 624
- "querySTSTokenTypeConfigurationDefaultProperties " on page 625
- "querySTSTokenTypeConfigurationCustomProperties " on page 626
- "setSTSDefaultTokenType " on page 626
- "updateSTSTokenTypeConfiguration " on page 627
- "removeSTSTokenTypeConfiguration" on page 628

Use the following commands to assign, unassign, and query endpoint configurations:

- "assignSTSEndpointTokenType " on page 628
- "listSTSAssignedEndpoints " on page 629
- "listSTSEndpointTokenTypes " on page 630
- "unassignSTSEndpointTokenType " on page 630
- "updateSTSEndpointTokenType " on page 631

Use the following commands to add, edit, delete, and list properties of the trust service:

- "addSTSProperty" on page 631
- "deleteSTSProperty" on page 632
- "editSTSProperty" on page 632
- "listSTSProperties" on page 633

Use the following command to force the trust service to reload your modified configuration without restarting the application server:

- "refreshSTS " on page 634

createSTSTokenTypeConfiguration

The createSTSTokenTypeConfiguration command is used to create a token provider configuration.

Target object

Specify the LocalName object, which is used as an identifier for the various configurations. The value for the LocalName object must be unique.

Required parameters

-URI
The URI of the token provider. This value must be unique across all configuration token type URIs. (String, required)

-HandlerFactory
Provide the fully qualified class name of an implementation of the `org.eclipse.higgins.sts.IObjectFactory` interface. (String, required)

Optional parameters

-lifetimeMinutes
Specifies the maximum lifetime to assign to an issued token provider. The default value is 120 minutes. (Integer, optional)

-distributedCache
Specifies whether to enable or disable distributed cache. Specify `true` to enable distributed cache capability. The default value is `false`. If you specify this option, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for this parameter for custom tokens. (Boolean, optional)

-tokenCacheFactory
Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter. (String, optional).

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.createSTSTokenTypeConfiguration('myTokenType', '[-HandlerFactory
 test.ibm.samples.myTokenType -URI http://ibm.com/tokens/schema/myTokenType]')
```

- Using Jython list:

```
AdminTask.createSTSTokenTypeConfiguration('myTokenType', ['-HandlerFactory',
 'test.ibm.samples.myTokenType', '-URI', 'http://ibm.com/tokens/schema/myTokenType'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.createSTSTokenTypeConfiguration('-interactive')
```

deleteSTSTokenTypeConfigurationCustomProperties

The deleteSTSTokenTypeConfigurationCustomProperties command is used to remove custom properties from a token provider configuration.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

-propertyNames
Specify the names of the custom properties to delete from the configuration. If any of the specified properties do not exist in your configuration, you will receive an error message. (String[], optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('myTokenType', '[-propertyNames
 com.ibm.ws.security.webChallengeIfCustomSubjectNotFound com.ibm.ws.security.defaultLoginConfig]')
```

- Using Jython list:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('myTokenType', ['-propertyNames',
 'com.ibm.ws.security.webChallengeIfCustomSubjectNotFound com.ibm.ws.security.defaultLoginConfig'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSTSTokenTypeConfigurationCustomProperties('-interactive')
```

listSTSConfiguredTokenTypes

The listSTSConfiguredTokenTypes command is used to list the local names of all configured token providers.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the local names of all configured token providers.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSConfiguredTokenTypes()
```

Interactive mode example usage

- Using Jython:

```
AdminTasklistSTSConfiguredTokenTypes('-interactive')
```

querySTSDefaultTokenType

The querySTSDefaultTokenType command is used to determine the local name of the default token provider.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the local name of the default token provider.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSDefaultTokenType()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSDefaultTokenType('-interactive')
```

querySTSTokenTypeConfigurationDefaultProperties

The querySTSTokenTypeConfigurationDefaultProperties command is used to query the trust service for the non-custom properties of a token provider.

Target object

Specify the LocalName object of the token provider to query.

Required parameters

None

Optional parameters

None

Return value

The command returns a `java.util.Properties` instance which contains the values of the non-custom properties. Non-custom properties include URI, HandlerFactory, lifetimeMinutes, distributedCache, postdatable, renewableAfterExpiration, and renewalWindowMinutes.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationDefaultProperties('-interactive')
```

querySTSTokenTypeConfigurationCustomProperties

The querySTSTokenTypeConfigurationCustomProperties command is used to query the trust service.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

None

Return value

The command returns a `java.util.Properties` instance containing the values of the custom properties.

Batch mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.querySTSTokenTypeConfigurationCustomProperties('-interactive')
```

setSTSDefaultTokenType

The setSTSDefaultTokenType command is used to set the default token provider for the trust service.

Target object

Specify the LocalName object of the token provider as default.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.setSTSDefaultTokenType('TokenType2')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.setSTSDefaultTokenType('-interactive')
```

updateSTSTokenTypeConfiguration

The updateSTSTokenTypeConfiguration command is used to update configuration data for a token provider. All parameters are optional. The parameters that are specified are updated in the configuration if the property already exists. If the property does not exist, it is added to the configuration. To remove custom properties, use the **deleteSTSTokenTypeConfigurationCustomProperties** command.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

-URI
The URI of the token provider. This value must be unique across all configuration token type URIs. (String, optional)

-HandlerFactory
Provide the fully qualified class name of an implementation of the `org.eclipse.higgins.sts.utilities.IObjectFactory` interface. (String, optional)

-lifetimeMinutes
The maximum lifetime to assign to an issued token provider. The default value is 120 minutes. (Integer, optional)

-distributedCache
Specifies whether to enable or disable distributed cache. Specify `true` to enable distributed cache capability. The default value is `false`. If you specify this option, the security context token provider generates a warning and modifies the WS-Security distributed cache configuration. Do not specify a value for this parameter for custom tokens. (Boolean, optional)

-postdatable
Set the value of this parameter to `true` to allow tokens of this token provider to be valid starting at a future time. The default value is false. (Boolean, optional)

-renewableAfterExpiration
Set the value of this parameter to `true` to allow tokens of this token provider to be renewable after expiration. The default value is false. (Boolean, optional)

-renewableWindowMinutes
Provide the number of minutes after a token has expired that a token of this token provider can be renewed. If this specified time has elapsed after expiration, then the token will no longer be available for renewal. The default value is 120 minutes. (Integer, optional)

-tokenCacheFactory
Specifies the fully qualified class name for the token provider. The secure conversation token handler class does not recognize this parameter. (String, optional).

-customProperties
Provide any additional custom properties. (java.util.Properties, optional).

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSTSTokenTypeConfiguration('myTokenType', '[-lifetimeMinutes 100
 -renewableAfterExpiration false -distributedCache true]')
```

- Using Jython list:

```
AdminTask.updateSTSTokenTypeConfiguration('myTokenType', ['-lifetimeMinutes', '100', '
 -renewableAfterExpiration', 'false', '-distributedCache', 'true'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSTSTokenTypeConfiguration('-interactive')
```

removeSTSTokenTypeConfiguration

The removeSTSTokenTypeConfiguration command removes a token provider configuration.

Target object

Specify the LocalName object of the token provider of interest.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.removeSTSTokenTypeConfiguration('myTokenType')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.removeSTSTokenTypeConfiguration ('-interactive')
```

assignSTSEndpointTokenType

The assignSTSEndpointTokenType command is used to give a token provider when a specific endpoint is accessed.

Target object

Specify the endpointURI object of the endpoint to assign a given token provider. If the specified endpoint has already been assigned a token provider, you will receive an error message.

Required parameters

None

Optional parameters

-LocalName
Specify the local name of the token provider to assign to the specified endpoint. If the token provider configuration does not exist, you will receive an error message. If this parameter is not specified, the default token provider is used. (String, optional)

-issuer
Specify the URI of the issuer that specifies the token provider to issue. This value can be null. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.assignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', '[-LocalName
 tokenType1]')
```

- Using Jython list:

```
AdminTask.assignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.assignSTSEndpointTokenType ('-interactive')
```

listSTSAssignedEndpoints

The listSTSAssignedEndpoints command is used to list the URIs of assigned endpoints.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns the URIs of all assigned endpoints.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSAssignedEndpoints()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSAssignedEndpoints ('-interactive')
```

listSTSEndpointTokenTypes

The listSTSEndpointTokenTypes command is used to query the Trust Service for the token provider assigned to a specific endpoint.

Target object

Specify the endpointURI object of the endpoint to query. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

None

Optional parameters

None

Return value

The command returns the local name of the token provider assigned to the specified endpoint.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSEndpointTokenTypes()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSEndpointTokenTypes ('-interactive')
```

unassignSTSEndpointTokenType

The unassignSTSEndpointTokenType command is used to unassign an endpoint from its token provider.

Target object

Specify the endpointURI object of the endpoint to unassign from a given token provider. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

-LocalName
Specify the local name of the token provider configuration to unassign from the specified endpoint. (String, required)

Optional parameters

-issuer
Specify the URI of the issuer in the token provider assignment to remove. (String, optional)

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.unassignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/','[-LocalName
 tokenType2]')
```

- Using Jython list:

```
AdminTask.unassignSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType2'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.unassignSTSEndpointTokenType ('-interactive')
```

updateSTSEndpointTokenType

The updateSTSEndpointTokenType command is used to assign a different token provider to a specified endpoint.

Target object

Specify the endpointURI object of the endpoint to update. An exception is raised if the specified endpoint has not been assigned a token provider.

Required parameters

-LocalName
Specify the local name of the token provider to assign to the specified endpoint. If the token provider configuration does not exist, you will receive an error message. If this parameter is not specified, the default token provider is used. (String, optional)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.updateSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', '[-LocalName
 tokenType2]')
```

- Using Jython list:

```
AdminTask.updateSTSEndpointTokenType('www.ibm.tokenservice/Ecommerce/', ['-LocalName',
 'tokenType2'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.updateSTSEndpointTokenType('-interactive')
```

addSTSProperty

The addSTSProperty command adds a new property for the trust service.

Target object

Specify a unique name for the new property (string, required).

Required parameters

-propertyValue
Specifies the value of the property to add. (String, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.addSTSProperty('pluginSCTVersion', '[-propertyValue 2.0]')
```

- Using Jython list:

```
AdminTask.addSTSProperty('pluginSCTVersion', ['-propertyValue', '2.0'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.addSTSProperty('-interactive')
```

deleteSTSProperty

The deleteSTSProperty command deletes an existing property from the trust service.

Target object

Specify the name of the property to delete.

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.deleteSTSProperty('pluginSCTVersion')
```

Interactive mode example usage

- Using Jython:

```
AdminTask.deleteSTSProperty('-interactive')
```

editSTSProperty

The editSTSProperty command modifies an existing property for the trust service.

Target object

Specify the name of the property to edit. (String, required)

Required parameters

-propertyValue
Specifies the new value for the property of interest. (String, required)

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython string:

```
AdminTask.editSTSProperty('pluginSCTVersion', '[-propertyValue 2.1]')
```

- Using Jython list:

```
AdminTask.editSTSProperty('pluginSCTVersion', ['-propertyValue', '2.1'])
```

Interactive mode example usage

- Using Jython:

```
AdminTask.editSTSProperty('-interactive')
```

listSTSProperties

The listSTSProperties command lists all existing properties and their corresponding values for the trust service.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a java.util.Properties instance that contains the names and values of the properties.

Batch mode example usage

- Using Jython:

```
AdminTask.listSTSProperties()
```

Interactive mode example usage

- Using Jython:

```
AdminTask.listSTSProperties('-interactive')
```

refreshSTS

The refreshSTS command refreshes your trust service configuration changes without restarting the application server.

Target object

None

Required parameters

None

Optional parameters

None

Return value

The command returns a success or failure message.

Batch mode example usage

- Using Jython:

```
AdminTask.refreshSTS()
```

Adding assured delivery to Web services through WS-ReliableMessaging

WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services reliable without having to write custom code. You can get different qualities of service with WS-ReliableMessaging. These range from protecting against loss of messages across a network, through to protecting against a server becoming unavailable.

About this task

With WebSphere Application Server, you can use WS-ReliableMessaging with Java API for XML-Based Web Services (JAX-WS) 2.1 Web services applications that use a SOAP over HTTP binding. To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent.

Note:

Support for the WS-ReliableMessaging standard was first introduced as part of the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services. At that time, the Reliable Asynchronous Messaging Profile (RAMP) Version 1.0 specification used WS-ReliableMessaging to ensure the reliable delivery of messages, and the Feature Pack for Web Services in WebSphere Application Server Version 6.1 included default policy sets that support this specification. You can migrate WebSphere Application Server Version 6.1 WS-ReliableMessaging configurations that use RAMP-based policy sets to the current version of the product.

Following on from the RAMP Version 1.0 specification, the Web Services Interoperability organization (WS-I) Reliable Secure Profile working group has developed Version 1.0 of an interoperability profile dealing with secure, reliable messaging capabilities for Web services. This

profile is similar to RAMP Version 1.0, except that it is updated to use WS-ReliableMessaging Version 1.1 with the OASIS WS-SecureConversation Version 1.3 specification. The WS-I RSP default policy sets provided in this version of WebSphere Application Server are an implementation of the Reliable Secure Profile Version 1.0 specification.

If you create JAX-WS based WS-Notification services, you can apply WS-ReliableMessaging policies to them to make your WS-Notification services reliable. For more information, see Configuring WS-Notification for reliable notification.

The WS-Policy implementation in WebSphere Application Server supports Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1. For more information, see “WS-Policy” on page 645.

To enable WS-ReliableMessaging for an application, you take the following broad actions:

1. Develop a Java API for XML-Based Web Services (JAX-WS) Web service provider or requester application.
2. Install the application into WebSphere Application Server.
3. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or Web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
4. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

For more information about using WS-ReliableMessaging, see the following topics:

- Learn about WS-ReliableMessaging.
- Configure endpoints to only support clients that use WS-ReliableMessaging.
- Build a reliable Web service application.
- Configure a WS-ReliableMessaging policy set.
- Attach and bind a WS-ReliableMessaging policy set to a Web service application.
- Detect and fix problems with WS-ReliableMessaging.

Learning about WS-ReliableMessaging

WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. Use these topics to learn more about WS-ReliableMessaging.

About this task

Without WS-ReliableMessaging, your Web services that require assured delivery of SOAP messages can either use a vendor-specific binding such as SOAP over JMS (which provides limited interoperability) or they can use SOAP over HTTP and rely upon you to write the associated durable message stores, custom retry logic at the sender, and duplicate detection at the receiver. With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services reliable without having to write custom code.

To enable WS-ReliableMessaging for an application, you take the following broad actions:

1. Develop a Java API for XML-Based Web Services (JAX-WS) Web service provider or requester application.

2. Install the application into WebSphere Application Server.
3. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or Web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
4. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

To learn about the WS-ReliableMessaging implementation in WebSphere Application Server, see the following topics:

WS-ReliableMessaging - How it works

WebSphere Application Server uses WS-ReliableMessaging as part of the transport layer for SOAP over HTTP messages. The message exchange patterns supported at the API layer are one-way "fire and forget", or two-way request and reply.

The reliability is provided by reliable messaging middleware that sits between the Web service requester and the Web service provider. This middleware layer is shown beneath the dotted line in the following diagram, and includes the reliable messaging source and the reliable messaging destination.

Note: When using WS-ReliableMessaging with a two-way programming API, if the requesting application fails and is restarted it will not receive its reply message. In this model, WS-ReliableMessaging is being used to protect from network failures only. Moreover:

- Client-side retransmissions only start after the client starts sending new messages to the service (this is true for both one-way and two-way operations).
- Two-way operations that resume cannot drive the response message right back to the client application; the message only gets back as far as the inbound sequence on the client.

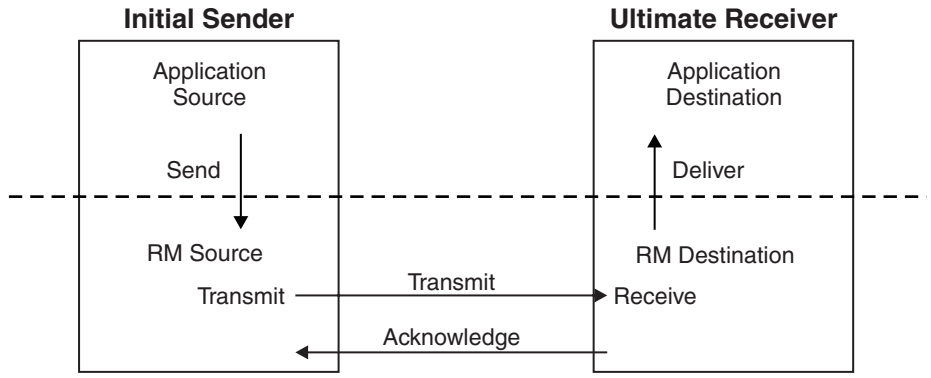


Figure 6. The interactions used to exchange Web services messages reliably.

In the previous diagram, the application source invokes a Web service. The sequence of interactions is as follows:

- The invocation is passed to the reliable messaging source.
- The reliable messaging source stores the message and then returns control to the application.
- The reliable messaging source sends the message to the reliable messaging destination.
- After the reliable messaging destination receives the message, it stores it locally and sends an acknowledgement message back to the reliable messaging source.
- The reliable messaging source can now delete its copy of the message.
- The reliable messaging destination can deliver the message to the application destination at any time after it receives it from the reliable messaging source.

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent.

The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus. For each attachment between an application and a policy set, you can select the bus and messaging engine to use for the reliable messaging protocol state.

Benefits of using WS-ReliableMessaging

With WS-ReliableMessaging, along with the other components of the Reliable Secure Profile, you can support your business-to-business Web services scenarios without having to write your own custom retry logic, duplicate detection code and persistence code.

WS-ReliableMessaging composes with other Web services standards as described in "WS-ReliableMessaging: supported specifications and standards" on page 643.

Qualities of service for WS-ReliableMessaging

You can get different qualities of service with WS-ReliableMessaging, depending on the level of durability and transaction support provided by the store used to manage the reliable messaging state. These qualities of service range from protecting against loss of messages across a network, through to protecting against server failure.

WebSphere Application Server provides the following three qualities of service for WS-ReliableMessaging using a SOAP over HTTP binding. All three qualities of service are supported when applications are deployed to the application server. Thin client and client container applications use the first option only.

Unmanaged non-persistent

You can configure Web service applications to use WS-ReliableMessaging with a default

in-memory store. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages.

Managed non-persistent
This in-memory quality of service option uses a messaging engine to manage the sequence state, and messages are written to disk if memory is low. This quality of service allows for the re-sending of messages that are lost in the network, and can also recover from server failure. However, state is discarded after a messaging engine restart so in this case you will lose messages.

Managed persistent
This quality of service for asynchronous Web service invocations is recoverable. This option also uses a messaging engine and message store to manage the sequence state. Messages are persisted at the Web service requester server and at the Web service provider server, and are recoverable if the server becomes unavailable. Messages that have not been successfully transmitted when a server becomes unavailable can continue to be transmitted after the server restarts.

Note:

- The quality of service you get when using WS-ReliableMessaging is a direct result of the durability of the store managing the messages.
- When you use in-order delivery and either of the managed qualities of service, if the service causes an error then the message is re-dispatched to the service.
- You must ensure that when interacting with other vendors implementations of WS-ReliableMessaging, the other implementations provide the quality of service you require.

How the different qualities of service are implemented

When the Web service application invokes the Web service, the SOAP message is added into the WS-ReliableMessaging store. For the Managed qualities of service, the sending application's transaction is used to put the message into the message store. After the transaction commits, the message is eligible for delivery. The other quality of service option is not transactional, so it considers the message eligible for delivery immediately.

The WS-ReliableMessaging protocol is used to reliably deliver the message to the target server where it is stored and acknowledged.

The message is read from the store and dispatched to the receiving application. For the Managed Persistent quality of service, a transaction is used to read the message and then dispatch the application.

For more information about using WS-ReliableMessaging transactions, see "Providing transactional recoverable messaging through WS-ReliableMessaging" on page 653.

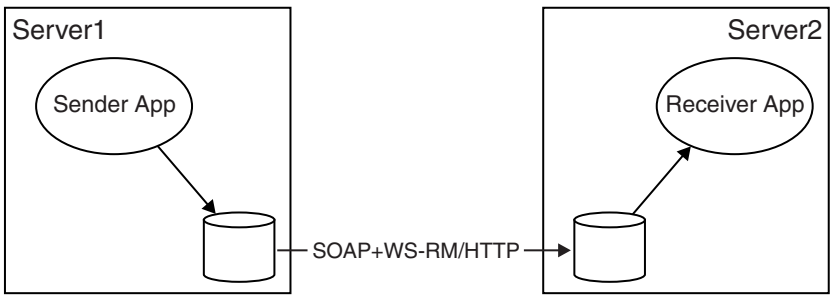


Figure 7. Using stores to exchange Web services messages reliably.

The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus. For each attachment between an application and a policy set, you can select the bus and messaging engine to use for the reliable messaging protocol state.

Use patterns for WS-ReliableMessaging

Links to descriptions of the use patterns that motivate WS-ReliableMessaging. Each use pattern description includes an overview of the business problem, of the technical solution without WS-ReliableMessaging, of the shortcomings of this solution, and of how you can use WS-ReliableMessaging to overcome these shortcomings.

Historically, most business-to-business integration has been implemented on a point-to-point basis. However this situation is rapidly changing and hub-and-spoke is becoming more important, particularly for supply chain use patterns. The point-to-point use pattern is also important because, although your eventual goal might be to implement a business-to-business hub, you might nonetheless begin with point-to-point prototypes and proofs of concept.

For a description of each use pattern, see the following topics:

- "Assured delivery for B2B Web services: point-to-point use pattern."
- "Assured delivery for B2B Web services: hub-and-spoke use pattern" on page 640.
- "Interoperation with other WS-ReliableMessaging providers: use pattern" on page 641.

Assured delivery for B2B Web services: point-to-point use pattern:

In this use pattern, a manufacturer sells its products through a network of affiliated dealerships. This manufacturer has initiated a pilot project to improve the IT integration between its own retail organization and half a dozen of the largest, most important dealerships.

The existing technical solution

Historically, business-to-business ″e-commerce″ has been conducted using Electronic Data Interchange (EDI). EDI is a set of standards for the content and formatting of business-to-business messages. For examples of these standards and messages, see the United Nations Directories for Electronic Data Interchange.

If the identities of communication partners are known and unchanging, the use of industry standard message definitions is not strictly necessary. Although other XML-based standards are available for conducting business-to-business e-commerce (such as the OASIS Electronic Business using eXtensible Markup Language (ebXML) specifications) the manufacturer has decided to investigate the use of Web services technologies, and is using WSDL documents from a variety of sources to define the service interfaces.

The interactions between the manufacturer and its dealers for the initial pilot project fall into two categories:

- Requests for information. The interaction is two-way, in that a request message is sent requesting some information, and a reply message is sent in the reverse direction containing the requested information. An example of a request for information going from a dealer to the manufacturer might be ″getOrderStatus″.
- Requests for update. These interactions are one-way, in that the sender of a request for update is not dependent on receiving a response in order to proceed with other work. An example of a request for update going from dealer to manufacturer might be ″placeOrder″. An example of a request for update going from manufacturer to dealer might be ″deliveryConfirmed″.

The manufacturer uses WebSphere Application Server to implement requests for information using SOAP over HTTP and SOAP over JMS. Dealers are free to choose their own implementation technology; they do not have to use WebSphere Application Server.

The manufacturer implements requests for update in two different ways:

- Using SOAP over HTTP. In this case the service is represented as a request and reply interaction that is considered to have succeeded when the requestor successfully receives a reply. The services have to be implemented to detect and successfully respond to duplicate requests (this is termed an idempotent operation), and the client has to be implemented to retry if the communication is interrupted after the request has been sent but before the reply has been received.
- To avoid the above limitations, the manufacturer also uses SOAP over JMS support from WebSphere Application Server and WebSphere MQ. In this case the request is represented as a one-way service, and the messages are delivered reliably. The manufacturer uses WebSphere MQ as the JMS Provider, and makes this solution available to all dealers that also use WebSphere Application Server and WebSphere MQ. It is not required that the dealer and manufacturer be connected in order for the message to be sent.

The messages are transmitted over Virtual Private Networks, to ensure the integrity and confidentiality of messages transmitted between the two businesses, and as a part of establishing the identity of the sender.

The business problem

Although both the manufacturer and its dealers are happy with the implementation of the request for information services, there are a number of issues in the request for update case:

- Using SOAP over HTTP:
  - For the manufacturer, implementing idempotent services is complicated and therefore more expensive in developer time. It increases the likelihood of coding errors, reducing the robustness of the solution and introducing the possibility of expensive dropped or duplicated orders.
  - For dealers, implementing the retry logic is similarly complex, expensive, and error-prone.
  - For both the manufacturer and the dealers, the requirement for both to be available in order to invoke the service is an issue. In particular, many dealers do not maintain seven-day availability of their systems, whereas for the manufacturer weekends are the ideal time to deliver price updates to the dealers. Similarly, being unable to place orders when connectivity between dealer and manufacturer is unavailable is a real business issue.
- Using SOAP over JMS:
  - Although requiring the use of WebSphere Application Server and WebSphere MQ is acceptable to the current collection of dealers, as the project expands there might be other partners who are unwilling or unable to use a common software platform.

The solution using WS-ReliableMessaging

With WS-ReliableMessaging support in WebSphere Application Server, the manufacturer can replace their existing custom-retry solutions for reliable one-way messaging with standard SOAP over HTTP one-way messaging. The removal of the retry logic from the application simplifies the application code, enabling simpler and quicker application development.

With WS-ReliableMessaging, the dealer and manufacturer do not need to be connected in order for the message to be sent.

The WS-ReliableMessaging standard adds reliability to SOAP over HTTP messaging, reducing the need to use SOAP over JMS.

Because WS-ReliableMessaging with SOAP over HTTP is an interoperable standard, the network of dealers need not use a common software platform.

Assured delivery for B2B Web services: hub-and-spoke use pattern:

In this use pattern, a manufacturer is looking for more than the ability to conduct transactions electronically with a fixed set of partners; the manufacturer needs a service that provides visibility to their inventory levels, so that the suppliers can manage their own product and inventory levels accordingly.

The manufacturer has many suppliers - around 2000 major suppliers and 250,000 other suppliers - and the set of supplier companies with which they do business is constantly changing.

The existing technical solution

The manufacturer currently has a custom-built supplier purchasing system. The system provides a trading hub into which suppliers can integrate, and a supplier portal that enables suppliers to continue with their (largely manual) existing processes. The interactions fall into three tiers:

- Tier one: XML message exchanges.
- Tier two: FTP and similar tools.
- Tier three: Fax.

Tiers one and two are conducted over a combination of leased lines and Virtual Private Networks (VPNs).

The business problem

The custom purchasing system has had disappointing take-up by the suppliers, and as a result the manufacturer continues to face large costs associated with the manual processes that it still uses with the vast majority of its suppliers. A major barrier to adoption of the new system by suppliers has been the cost of integrating supplier systems with the trading hub.

The current solution also has costs for the manufacturers. In particular, the use of leased lines is expensive, and the use of VPNs is difficult to manage when scaled to a large number of suppliers.

The solution using WS-ReliableMessaging

Describing the services using WSDL, and using Web services standards to implement them, reduces the costs incurred by the manufacturer and suppliers:

- The prevalence of support for Web services amongst diverse software vendors makes it easier and cheaper for suppliers to exploit this technology.
- The familiarity of the developer community with Web services technologies (and WSDL in particular), and the rich tool support, means that the use of WSDL-described messages make the message schemas easier to use in the context of Web services.
- WS-ReliableMessaging is used to ensure that messages are reliably delivered, and duplicate messages are eliminated.
- Web services security technologies enable secure interactions across the Internet, without requiring leased lines and VPNs.

Interoperation with other WS-ReliableMessaging providers: use pattern:

Web services enable interoperability between heterogeneous platforms. This requirement arises whenever an organization finds itself with applications on one platform (for example WebSphere Application Server) that need to work with applications on another platform, whether as a result of merger and acquisition activity, of a deliberate multi-vendor strategy, or as a result of independent software purchasing decisions taken in different parts of the business.

The existing technical solution

A variety of technical solutions exist for application-to-application integration between WebSphere Application Server and other environments. Most of these involve the use of additional third-party or IBM software to facilitate the integration.

More recently, the introduction of Web services support has made interoperability possible without the use of additional components.

The business problem

Basic Web services support (using SOAP over HTTP) does enable interoperability, but has the following limitations:

- Reliability: The absence of a WS-ReliableMessaging implementation means that the application logic needs to be extended to handle lost or duplicated messages.
- Flexibility: The absence of asynchronous support for Web services means that support is limited to synchronous interactions.

Note: Although both request and reply and one-way messaging are supported in earlier version of WebSphere Application Server, they were implemented in a synchronous fashion. This meant that when a Web services client invoked a service it did not receive control back from the middleware until after the service application endpoint had been invoked.

The absence of asynchronous, reliable support for Web services often leads you to use one of the other approaches, involving additional components. The additional components often use proprietary communication channels or APIs.

The solution using WS-ReliableMessaging

The addition of WS-ReliableMessaging support to WebSphere Application Server and to other environments enables you to develop reliable asynchronous Web services on both platforms. These services should interoperate without additional IBM or third-party components or proprietary bindings.

WS-ReliableMessaging sequences

The WS-ReliableMessaging protocol relies on a “sequence” to manage the transmission of messages from reliable messaging source to reliable messaging destination. Each application message is given an identifier that identifies both the sequence and the message number (the position) within the sequence. Protocol flows are used to create sequences, to acknowledge messages and to terminate sequences.

You can think of a sequence as being a structured conversation between the reliable source and the reliable destination, through which each message in the sequence is passed reliably. A sequence also passes on the set of messages in the sequence in the order in which it receives them, so if it is important that messages are processed in a particular order - for example, if money must be credited to a bank account before a debit instruction is received to pay for a purchase - then those messages should be included in the same sequence.

The developer of a reliable Web services application does not need to be aware of sequences, but the system administrator needs to monitor and manage sequences, as described in “Detecting and fixing problems with WS-ReliableMessaging” on page 661.

WS-ReliableMessaging - terminology

Foreign destination
A foreign destination is a software agent, outside of the system, that receives messages reliably using WS-ReliableMessaging.

Foreign source
A Foreign source is a software agent, outside of the system, that sends messages reliably using WS-ReliableMessaging.

Implementation WSDL
A WSDL document that describes not only the interface to a service (that is the messages, port

types and bindings) but also the service implementation (that is, it has service and port elements). The service interface should preferably be defined by importing a separate *interface WSDL* document.

Interface WSDL
A WSDL document that describes only the interface to a service. That is, it defines messages, port types and bindings, but not services and ports.

Policy set instance document
A document containing configuration details for a selection of Web services standards. For more information, see Securing Web services applications with policy sets using the administrative console

Provider application
A provider application is an application that implements a service.

Reliable Web service provider
See also the general definition of a *provider application*. The Reliable Web service provider is a provider application that has been deployed into the system being modeled, and is configured to use WS-ReliableMessaging. This is the WebSphere Application Server-hosted equivalent of a *foreign destination*. It is outside of the boundary of the system and interacts with the system.

Reliable Web service requester
See also the general definition of a *requester application*. The Reliable Web service requester is a requester application that has been deployed into the system being modeled, and is configured to use WS-ReliableMessaging. This is the WebSphere Application Server-hosted equivalent of a *foreign source*. It is outside of the boundary of the system and interacts with the system.

Requester application
A requester application is an application that makes Web service requests.

Terminate (a sequence)
When the reliable messaging source has completed its use of a sequence, it sends a TerminateSequence message to the reliable messaging destination to indicate that the sequence is complete and that it will not be sending any further messages related to the sequence. The reliable messaging destination can then safely reclaim any resources associated with the sequence.

WS-ReliableMessaging: supported specifications and standards

WebSphere Application Server provides support for two levels of the WS-ReliableMessaging specification. This enables backward compatibility with vendors that provide WS-ReliableMessaging support at the February 2005 level, as well as meeting the requirements of the current OASIS specification. This implementation of WS-ReliableMessaging also composes with many other Web services standards.

Details of the supported WS-ReliableMessaging specifications are available at the following Web addresses:

- The WS-ReliableMessaging specification Version 1.0, February 2005.
- The OASIS WS-ReliableMessaging specification Version 1.1, February 2007.

Support for the WS-ReliableMessaging standard was first introduced as part of the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services. At that time, the Reliable Asynchronous Messaging Profile (RAMP) Version 1.0 specification used WS-ReliableMessaging to ensure the reliable delivery of messages, and the Feature Pack for Web Services in WebSphere Application Server Version 6.1 included default policy sets that support this specification. You can migrate WebSphere Application Server Version 6.1 WS-ReliableMessaging configurations that use RAMP-based policy sets to the current version of the product.

Following on from the RAMP Version 1.0 specification, the Web Services Interoperability organization (WS-I) Reliable Secure Profile working group has developed Version 1.0 of an interoperability profile

dealing with secure, reliable messaging capabilities for Web services. This profile is similar to RAMP Version 1.0, except that it is updated to use WS-ReliableMessaging Version 1.1 with the OASIS WS-SecureConversation Version 1.3 specification. The WS-I RSP default policy sets provided in this version of WebSphere Application Server are an implementation of the Reliable Secure Profile Version 1.0 specification.

The extent to which WS-ReliableMessaging composes with other Web services standards is described in the following sections:

- "WS-Addressing"
- "WS-AtomicTransactions"
- "WS-MakeConnection"
- "WS-Notification" on page 645
- "WS-Policy" on page 645
- "WS-SecureConversation" on page 645
- "WS-Security" on page 645

WS-Addressing

The WS-ReliableMessaging specification uses WS-Addressing, and the implementation fully supports the asynchronous request and reply model given in the WS-Addressing specification.

Note: WS-ReliableMessaging Version 1.1 messaging requires WS-Addressing to be mandatory. If you use a policy set that includes WS-ReliableMessaging and WS-Addressing policies, and the WS-Addressing policy is configured as optional, then WebSphere Application Server overrides the WS-Addressing setting and automatically enables WS-Addressing.

WS-AtomicTransactions

WS-ReliableMessaging transactions do not use the WS-AtomicTransactions protocol. The relationship between these two protocols is as follows:

- WS-AtomicTransactions and WS-ReliableMessaging are mutually exclusive when WS-ReliableMessaging is being used, with a managed store, to provide transactional recoverable messaging.
- If WS-ReliableMessaging is configured to use an in-memory store, then there are cases where a WS-AtomicTransaction can be flowed between the reliable messaging source and the reliable messaging destination for two-way invocations. In this situation, WS-ReliableMessaging only protects against network failures, not against server failure.

For more information about WS-AtomicTransactions, see Transaction support in WebSphere Application Server. For more information about using WS-ReliableMessaging transactions, see "Providing transactional recoverable messaging through WS-ReliableMessaging" on page 653.

WS-MakeConnection

WS-ReliableMessaging Version 1.1 uses the WS-MakeConnection protocol to enable synchronous message exchange. For more information about this protocol, see the WS-MakeConnection specification Version 1.1, February 28 2008.

WS-MakeConnection uses information contained in WS-Addressing message headers, so for any application that uses reliable synchronous message exchange you must include both WS-ReliableMessaging and WS-Addressing policy in the policy set.

WS-Notification

If you create JAX-WS based WS-Notification services, you can apply WS-ReliableMessaging policies to them to make your WS-Notification services reliable. For more information, see Configuring WS-Notification for reliable notification.

Note: In this release, there are two types of WS-Notification service:

- **Version 7.0**: You configure a Version 7.0 WS-Notification service and service points if you want to compose a JAX-WS WS-Notification service with WS-ReliableMessaging, or if you want to apply JAX-WS handlers to your WS-Notification service. This is the recommended type of service for new deployments.
- **Version 6.1**: You configure a Version 6.1 WS-Notification service and service points if you want to expose a JAX-RPC WS-Notification service using the same technology provided in WebSphere Application Server Version 6.1, including the ability to apply JAX-RPC handlers to the service.

WS-Policy

The WS-Policy implementation in WebSphere Application Server supports Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1.

You can use the WS-Policy protocol to exchange policies in standard format. You can communicate the policy configuration to any other client, service registry or service that supports the WS-Policy specification, including non-WebSphere Application Server products in a heterogeneous environment. For a service provider, the policy configuration can be shared in published WSDL. For a client, the client can obtain the policy of the service provider in the standard WS-PolicyAttachments format and use this information to establish a configuration that is acceptable to both the client and the service provider. In other words, the client can be configured dynamically, based on the policies supported by its service provider.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

WS-SecureConversation

WS-ReliableMessaging is designed to work with WS-SecureConversation. A secure conversation context is established and this is used to secure the application messages and the WS-ReliableMessaging protocol messages.

To use WS-SecureConversation, create or apply a policy set that includes both WS-ReliableMessaging and WS-SecureConversation. For example, either of the WS-I RSP default policy sets.

WS-Security

WS-ReliableMessaging composes with WS-Security. The WS-ReliableMessaging headers appended to application messages are signed if required. The WS-ReliableMessaging protocol messages are signed and encrypted if required.

Security processing is done close to the transport: after WS-ReliableMessaging processing at the Web service requester and before WS-ReliableMessaging processing at the Web service provider. This means the messages held in the WS-ReliableMessaging store are not signed and encrypted, so the emphasis is on the administrator to secure the store, if the store being used is the messaging engine in a service integration bus.

Note: If possible, use WS-SecureConversation rather than WS-Security because the WS-SecureConversation protocol is less susceptible to security attacks.

Configuring endpoints to only support clients that use WS-ReliableMessaging

By default, when a WS-ReliableMessaging enabled policy set is attached to an endpoint, the server supports clients that use reliable messaging and clients that do not use reliable messaging. In this version of the product, you can configure endpoints to only support clients that use reliable messaging.

About this task

You configure endpoints to only support clients that use reliable messaging by setting a property in either of the following ways:

- Set a property when packaging the application.
- Set a property as a JVM argument for the server.

This setting is reflected by WS-Policy if engaged. For information about how to engage WS-Policy, see "Using WS-Policy to exchange policies in a standard format" on page 714.

- When packaging the application, configure endpoints to only support clients that use reliable messaging by setting the **strictlyEnforceWSRM** property in the META-INF/MANIFEST.MF of a WAR file or EJB module.
- Using a JVM argument for the server, configure endpoints to only support clients that use reliable messaging by defining the Java virtual machine custom property **com.ibm.ws.websvcs.rm.strictlyEnforceWSRM** on the server. For more information, see Configuring the JVM.

Building a reliable Web service application

Develop a Java API for XML-Based Web Services (JAX-WS) provider or requester application, and configure a policy set to enable WS-ReliableMessaging. Install your application then attach the policy set. If you want to use either of the managed qualities of service, bind the application or policy set to a service integration bus and messaging engine.

Before you begin

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

About this task

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus.For each attachment between an application and a policy set, you can select the bus and messaging engine to use for the reliable messaging protocol state.

To enable WS-ReliableMessaging for an application, you take the following broad actions:

1. Develop a Java API for XML-Based Web Services (JAX-WS) Web service provider or requester application.
2. Install the application into WebSphere Application Server.

3. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or Web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
4. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

1. Develop your JAX-WS Web service application.

   For a Web service requester application that sends messages reliably:

   a. Get an implementation WSDL document, and select the SOAP over HTTP binding. The WSDL should be WS-I Basic Profile compliant.
   b. Build the JAX-WS application from the WSDL implementation document.
   c. (Optional) Enable transaction support for outbound (requester) one-way message sends. For more information, see “Providing transactional recoverable messaging through WS-ReliableMessaging” on page 653.
   d. (Optional) Use the **waitUntilSequenceCompleted** method on the sequenceManager to ensure that reliable messaging state is released after the client finishes messaging, as described in “Example: Code for waiting for a sequence to complete” on page 652.
   e. (Optional) If you want to use in-order delivery (that is, you want WS-ReliableMessaging to make messages available to your requester application in the order that they were sent), then you must also configure your requester application to poll for the messages in the order in which it wants to receive them. For more information, see “Configuring the WS-ReliableMessaging policy” on page 907.

   For a Web service provider application that requires reliable messaging:

   a. Write or get an interface WSDL document that describes the service interface. The document should be compliant with the WS-I Basic Profile.
   b. Write or get an implementation WSDL document, and select the SOAP over HTTP binding. The WSDL should remain WS-I Basic Profile compliant.
   c. Build the JAX-WS application from the WSDL implementation document.

   Your client application can also take programmatic control of WS-ReliableMessaging sequences. This helps manage resources on the server, for example by removing sequences after a client application has finished messaging. You can add code to create sequences, send acknowledgement requests, close sequences, terminate sequences and wait until sequences are complete. For more information, including example code, see “Controlling WS-ReliableMessaging sequences programmatically.”
2. Configure a policy set instance to enable WS-ReliableMessaging.
3. Install your reliable JAX-WS Web service application.
4. Attach and bind a WS-ReliableMessaging policy set to your application.
5. Save your changes to the master configuration.
6. Stop then restart the server.

Results

A reliable JAX-WS application is deployed into a suitably configured environment and started.

Controlling WS-ReliableMessaging sequences programmatically

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. This helps manage resources on the server, for example by removing sequences after a client application has finished messaging. You can add code to create sequences, send acknowledgement requests, close sequences, terminate sequences and wait until sequences are complete.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

About this task

By closing sequences programmatically, you limit the number of open sequences a single client has to support in a single JVM at one time.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. Use the following code fragment to achieve this:

```
import com.ibm.wsspi.wsrm.WSRMSequenceManager;
import com.ibm.wsspi.wsrm.WSRMSequenceManagerFactory;

.........

     // Get the factory
     WSRMSequenceManagerFactory factory = WSRMSequenceManagerFactory
         .getInstance();

     // Get the sequence manager instance
     WSRMSequenceManager sequenceManager = factory.createWSRMSequenceManager();
```

All **WSRMSequenceManager** methods take the following parameters:

- The client instance object. This is either a **Dispatch client** instance, or the **Dynamic proxy client**. For details of the client types, see JAX-WS client programming model.
- 2) The **Port QName** instance for the target endpoint.

To control WS-ReliableMessaging sequences programmatically, add code to your client application as described in the following steps:

- Add code to create a sequence.
- Add code to send an acknowledgement request.
- Add code to close a sequence.
- Add code to terminate a sequence.
- Add code to wait for a sequence to complete.

Example: Code for creating a sequence:

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. Use these code fragments as guidance for coding your reliable messaging client application to create a sequence.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. The first of the following code examples includes code to create a WSRMSequenceManager object. If you need more information about working with this object, see “Controlling WS-ReliableMessaging sequences programmatically” on page 647.

Example code

To create a new reliable messaging sequence, first create a **WSRMSequenceProperties** object:

```
import com.ibm.wsspi.wsrm.WSRMSequenceProperties

.........

WSRMSequenceManager sequenceManager = WSRMSequenceManagerFactory
         .getInstance().createWSRMSequenceManager();

WSRMSequenceProperties sequenceProperties = sequenceManager
         .createNewWSRMSequenceProperties();
```

To set the available properties use the following methods:

```
  /**
   * Sets the target provider endpoint.
   * A null value will cause a NullPointerException when the WSRMSequenceProperties object is used.
   *
   * @param providerEndPoint The target service endpoint URI
   */
  public void setTargetEndpointUri(String providerEndPoint);

  /**
   * This is used to indicate that a response flow is required between the provider and requester and the response
   * flow will be established at create sequence time
   *
   * By calling this method it will indicate that a response flow is required.
   */
  public void setUseOfferedSequenceId();

  /**
   * Set the Soap version for RM protocol messages.
   * The default value for this property is WSRMSequenceProperties.SOAP_11
   *
   * @param soapVersion
   */
  public void setSoapVersion(int soapVersion);

  /**
   * If the Sequence Acknowledgement messages are to be sent back asynchronously call this method.
   *
   */
  public void useAsyncTransport();
```

To create the reliable messaging sequence use the **createNewWSRMSequence** method on the **WSRMSequenceManager**:

```
  /**
   * Initiates a new sequence handshake between this client and the target EPR specified in the
   * WSRMSequenceProperties instance.
   *
   * This sequence will only be valid for the client issuing the createNewWSRMSequence call.
   *
   * When returning from this call, there is no guarantee that the sequence has been established.
   *
   * @throws NullPointerException if the sequenceProperties object is null, or the target EPR is null
   *
   * @param clientObject The JAX-WS Dispatch instance, or the Dynamic Proxy client instance.
   * @param sequencePropeties The properties for creating the reliable messaging sequence
   * @throws WSRMNotEnabledException
   * @throws WSRMSequenceAlreadyExistsException
   */
```

```
public void createNewWSRMSequence(Object clientObject, QName portQName, WSRMSequenceProperties sequencePropeties)

throws WSRMNotEnabledException,
       WSRMSequenceAlreadyExistsException;
```

Example: Code for sending an acknowledgement request:

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. Use these code fragments as guidance for coding your reliable messaging client application to send an acknowledgement request.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. For information and example code explaining how to achieve this, see “Controlling WS-ReliableMessaging sequences programmatically” on page 647.

For a more complete specification of sending an acknowledgement request, see the “WS-ReliableMessaging: supported specifications and standards” on page 643.

Example code

To send an acknowledgement request for a WS-ReliableMessaging sequence, use the following method on the **WSRMSequenceManager**:

```
/**
 * Sending an acknowledgement request sends the ACK requested message to the specified target endPointUri.
 * The target will respond with a range of messages that can be acknowledged for the current reliable messaging
 * sequence.
 *
 * @param clientObject The JAX-WS Dispatch instance, or the Dynamic Proxy client instance.
 * @param portQName
 * @param endPointUri The target endpoint uri
 * @throws WSRMNotEnabledException
 * @throws WSRMSequenceUnknownException
 * @throws WSRMSequenceTerminatedException
 * @throws WSRMSequenceClosedException
 */
public void sendAcknowledgementRequest(Object clientObject, QName portQName, String endPointUri)

throws WSRMNotEnabledException,
       WSRMSequenceUnknownException,
       WSRMSequenceTerminatedException,
       WSRMSequenceClosedException;
```

Example: Code for closing a sequence:

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. Use these code fragments as guidance for coding your reliable messaging client application to close a sequence.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. For information and example code explaining how to achieve this, see “Controlling WS-ReliableMessaging sequences programmatically” on page 647.

For a more complete specification of closing a sequence, see the “WS-ReliableMessaging: supported specifications and standards” on page 643.

By closing sequences programmatically, you limit the number of open sequences a single client has to support in a single JVM at one time.

Example code

To close a WS-ReliableMessaging sequence use the following method on the **WSRMSequenceManager**:

```
/**
 * Closes the Web services reliable messaging session from this application to
 * the endpoint url specified.
 *
 * Throws a WSRMSequenceTerminatedException if the session between this application
 * and the target endpoint url is already closed
 *
 * Throws a WSRMSequenceTerminatedException when the session between this application
 * and the target endpoint has already been terminated.
 *
 * Throws WSRMSequenceUnknownException exception when either reliable messaging is not engaged to
 * the specified endpoint url or the sequence has previously been terminated and removed.
 *
 * @param clientObject The JAX-WS Dispatch instance, or the Dynamic Proxy client instance.
 * @param endPointUri The target endpoint url
 *
 * @throws WSRMNotEnabledException
 * @throws WSRMSequenceUnknownException
 * @throws WSRMSequenceClosedException
 * @throws WSRMSequenceTerminatedException
 */
public void closeSequence(Object clientObject, QName portQName, String endPointUri)

throws WSRMNotEnabledException,
       WSRMSequenceUnknownException,
       WSRMSequenceClosedException,
       WSRMSequenceTerminatedException;
```

Example: Code for terminating a sequence:

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. Use these code fragments as guidance for coding your reliable messaging client application to terminate a sequence.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. For information and example code explaining how to achieve this, see “Controlling WS-ReliableMessaging sequences programmatically” on page 647.

For a more complete specification of terminating a sequence, see the “WS-ReliableMessaging: supported specifications and standards” on page 643.

By closing sequences programmatically, you limit the number of open sequences a single client has to support in a single JVM at one time.

Example code

To terminate a WS-ReliableMessaging sequence use the following method on the **WSRMSequenceManager**:

```
/**
 * Terminates Web services reliable messaging session from this application to
 * the endpoint url specified.
 *
 * Throws a WSRMSequenceTerminatedException when the session between this application
 * and the target endpoint has already been terminated.
 *
 * Throws WSRMSequenceUnknownException exception when either reliable messaging is not engaged to
 * the specified endpoint url or the sequence has previously been terminated and removed.
 *
 * @param clientObject The JAX-WS Dispatch instance, or the Dynamic Proxy client instance.
 * @param endPointUri The target endpoint url
 * @throws WSRMNotEnabledException
 *
 * @throws WSRMSequenceTerminatedException
 * @throws WSRMSequenceUnknownException
 */
public void terminateSequence(Object clientObject, QName portQName, String endPointUri) throws WSRMNotEnabledException;
```

Example: Code for waiting for a sequence to complete:

Your client application can use the WSRMSequenceManager, part of the WebSphere Application Server SPI for reliable messaging, to gain programmatic control over reliable messaging sequences. Use the **waitUntilSequenceCompleted** method on the sequenceManager to ensure that reliable messaging state is released after the client finishes messaging.

Before you begin

The WebSphere Application Server SPI for reliable messaging always uses the static policy set configuration that is applied to the client from which the SPI is called. It does not use any alternative policy set that is subsequently configured by WS-Policy to meet the requirements of a WS-Policy intersection.

For your client application to gain programmatic control over reliable messaging sequences, it needs access to a WSRMSequenceManager instance. For information and example code explaining how to achieve this, see “Controlling WS-ReliableMessaging sequences programmatically” on page 647.

Example code

To wait for a reliable messaging sequence to complete, you use a method call that ensures that all messages have been sent and acknowledged by the target service. After the sequence is completed, it is terminated and cleaned up. By closing sequences programmatically, you limit the number of open sequences a single client has to support in a single JVM at one time.

There are two ways of using the **waitUntilSequenceCompleted** method:

- `public boolean waitUntilSequenceCompleted(Object clientObject, QName portQName, String endPointUri, long waitTime)`

  This method call waits for the specified **waitTime** for the reliable messaging sequence to complete. If the sequence does not complete in the specified time, the method returns false. If the sequence does complete in time, the method returns true.
- `public boolean waitUntilSequenceCompleted(Object clientObject, QName portQName, String endPointUri)`

  This method call does not return until the reliable messaging sequence is completed.

Providing transactional recoverable messaging through WS-ReliableMessaging

If your WS-ReliableMessaging application is running inside the Web container and using a managed quality of service, you can use WS-ReliableMessaging to provide transactional recoverable messaging.

About this task

The WS-ReliableMessaging transactional model is as follows:

- On the Web service requester side, the transaction is between the application and the local managed store.
- The WS-ReliableMessaging protocol delivers the message to the Web service provider side, where a different transaction is used between the second managed store and the application being dispatched.

For the outbound (requestor) case on a one-way message send, if the **enableTransactionalOneWay** property is set to true, then the send is performed under any transactional context currently held by the application's thread. (Note that transactions are not supported under an outbound two-way message exchange).

For the inbound (provider) case, if the **inOrderDelivery** property is set to true, then an inbound message is dispatched to the application under a transaction. In the case of an inbound two way message exchange, the response is also generated under that transaction and is not sent until that transaction has committed.

Note:

WS-ReliableMessaging transactions do not use the WS-AtomicTransactions protocol. The relationship between these two protocols is as follows:

- WS-AtomicTransactions and WS-ReliableMessaging are mutually exclusive when WS-ReliableMessaging is being used, with a managed store, to provide transactional recoverable messaging.
- If WS-ReliableMessaging is configured to use an in-memory store, then there are cases where a WS-AtomicTransaction can be flowed between the reliable messaging source and the reliable messaging destination for two-way invocations. In this situation, WS-ReliableMessaging only protects against network failures, not against server failure.

For more information, see "WS-AtomicTransactions" on page 644.

To provide transactional recoverable messaging through WS-ReliableMessaging, work through the steps described in "Building a reliable Web service application" on page 646 and also complete the following additional steps:

- To enable transactional messaging for outbound (requester) one-way message sends, when you develop your JAX-WS Web service application set the **enableTransactionalOneWay** property to Boolean.TRUE (or the string "true") in the jaxWS request context map.
- To enable transactional messaging for inbound (provider) one-way and two-way message exchanges, when you configure your WS-ReliableMessaging policy either use the administrative console to select the option **Deliver messages in the order that they were sent** or use the wsadmin tool to set the **inOrderDelivery** property to "true".

Configuring a WS-ReliableMessaging policy set using the administrative console

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. Use the administrative console to configure a policy set for reliable messaging.

Before you begin

You can configure a reliable messaging policy set using the administrative console as described in this task, or you can configure a reliable messaging policy set using the wsadmin tool.

The following default policy sets work with WS-ReliableMessaging applications:

- WS-I RSP
- WS-I RSP ND
- LTPA WS-I RSP
- Username WS-I RSP
- WSReliableMessaging 1_0
- WSReliableMessaging default
- WSReliableMessaging persistent

For more information, see “WS-ReliableMessaging default policy sets” on page 811.

If you can use any of these default policy sets without needing to modify their configuration, you need not complete this task. You are ready to attach and bind the default policy set to your application.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

About this task

To configure a reliable messaging policy set using the administrative console, complete the following steps:

1. Create a policy set. You can create a new policy set, or copy and rename an existing policy set - either one that you have previously created, or one of the WS-ReliableMessaging default policy sets.
2. Check that your policy set includes the policy types **WS-ReliableMessaging** and **WS-Addressing**. Add these policy types if necessary. These policy types contain the configuration options that support WS-ReliableMessaging. WS-Addressing provides the asynchronous request and reply capabilities for WS-ReliableMessaging, and is also required for WS-ReliableMessaging Version 1.1 synchronous messaging.

   Note:

   - If you want to use secure conversation and reliable messaging policies in the same policy set, the secure conversation bindings must be configured to require that the reliable messaging headers are signed. The reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND) are specifically designed and configured to use secure conversation and reliable messaging in the same policy set. If you use a copy of one of the reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND), no further configuration of the secure conversation bindings is required. Otherwise, see “Configuring WS-SecureConversation to work with WS-ReliableMessaging” on page 656.
   - WS-ReliableMessaging Version 1.1 messaging requires WS-Addressing to be mandatory. If you use a policy set that includes WS-ReliableMessaging and WS-Addressing policies, and the WS-Addressing policy is configured as optional, then WebSphere Application Server overrides the WS-Addressing setting and automatically enables WS-Addressing.
3. Configure the **WS-ReliableMessaging** policy type attributes. For the WS-ReliableMessaging policy you can configure the version of the WS-ReliableMessaging standard that you want to use, the order in which messages are delivered, and the required quality of service (the reliability level) for message delivery.

Note: In WebSphere Application Server Version 6.1, you could also configure whether or not to use the WS-MakeConnection protocol. This configuration option has now been removed from the administrative console panel, because the product now automatically determines whether WS-MakeConnection is used, based on the following criteria:

- Whether you are using WS-ReliableMessaging Version 1.0 or Version 1.1.
- Whether the requester supports WS-MakeConnection.
- Whether the message exchange protocol is synchronous or asynchronous.

4. Save your changes to the master configuration.

What to do next

You are now ready to attach your application to the policy set and define the bindings that you want to use.

Configuring a WS-ReliableMessaging policy set using the wsadmin tool

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. Use command scripts to configure a policy set for reliable messaging.

Before you begin

You can configure a reliable messaging policy set using the wsadmin tool as described in this task, or you can configure a reliable messaging policy set using the administrative console.

The following default policy sets work with WS-ReliableMessaging applications:

- WS-I RSP
- WS-I RSP ND
- LTPA WS-I RSP
- Username WS-I RSP
- WSReliableMessaging 1_0
- WSReliableMessaging default
- WSReliableMessaging persistent

For more information, see “WS-ReliableMessaging default policy sets” on page 811.

If you can use any of these default policy sets without needing to modify their configuration, you need not complete this task. You are ready to attach your application to the default policy set and define the bindings that you want to use.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

About this task

To configure a reliable messaging policy set using the wsadmin tool, complete the following steps:

1. Create a policy set. Use the createPolicySet command to create a new policy set, or the **copyPolicySet** command to copy and rename an existing policy set - either one that you have previously created, or one of the two “WS-ReliableMessaging default policy sets” on page 811. For more information, see Creating and copying policy sets using the wsadmin tool.

2. If the policy set does not include both the policy types **WSReliableMessaging** and **WSAddressing**, add these policy types using the addPolicyType command as described in Creating and copying policy sets using the wsadmin tool. For example:

```
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType WSReliableMessaging]')
AdminTask.addPolicyType('[-policySet PolicySet1 -policyType WSAddressing]')
```

   These policy types contain the configuration options that support WS-ReliableMessaging. WS-Addressing provides the asynchronous request and reply capabilities for WS-ReliableMessaging, and is also required for WS-ReliableMessaging Version 1.1 synchronous messaging.

   Note:

   - If you want to use secure conversation and reliable messaging policies in the same policy set, the secure conversation bindings must be configured to require that the reliable messaging headers are signed. The reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND) are specifically designed and configured to use secure conversation and reliable messaging in the same policy set. If you use a copy of one of the reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND), no further configuration of the secure conversation bindings is required. Otherwise, see "Configuring WS-SecureConversation to work with WS-ReliableMessaging."
   - WS-ReliableMessaging Version 1.1 messaging requires WS-Addressing to be mandatory. If you use a policy set that includes WS-ReliableMessaging and WS-Addressing policies, and the WS-Addressing policy is configured as optional, then WebSphere Application Server overrides the WS-Addressing setting and automatically enables WS-Addressing.

3. Configure the **WS-ReliableMessaging** policy type attributes.

   For the WS-ReliableMessaging policy you can configure the version of the WS-ReliableMessaging standard that you want to use, the order in which messages are delivered, and the required quality of service (the reliability level) for message delivery. For detailed information about these configurable attributes, see "WS-ReliableMessaging settings" on page 908.

   Use the setPolicyType command to configure these attributes. For example:

```
AdminTask.setPolicyType('-policySet PolicySet1 -policyType WSReliableMessaging -attributes "[[inOrderDelivery false][specLevel 1.0]
[enabled true][qualityOfService managedPersistent][type WSReliableMessaging]]" -replace'
```

4. Save your changes to the master configuration.

   To save your configuration changes, enter the following command:

   ```
   AdminConfig.save()
   ```

What to do next

You are now ready to attach your application to the default policy set and define the bindings that you want to use.

Configuring WS-SecureConversation to work with WS-ReliableMessaging

Configure secure conversation to expect the reliable messaging headers to be signed, and to ensure that the scoping security context token does not expire before reliable messaging recovers and resends persistent messages.

- "Configure secure conversation to expect the reliable messaging headers to be signed"
- "Configure secure conversation to increase the timeout setting for the scoping security context token" on page 657

Configure secure conversation to expect the reliable messaging headers to be signed:

About this task

Although secure conversation allows message headers to remain unsigned, the reliable messaging policy requires the reliable messaging headers to be signed. If you want to use secure conversation and reliable messaging policies in the same policy set, the secure conversation bindings must be configured to require

that the reliable messaging headers are signed. To achieve this, complete one of the following steps:

- Create your policy set from a copy of one of the reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND). This policy set contains instances of the secure conversation and reliable messaging policies that are configured to work together.
- Create your policy set from a copy of the secure conversation policy set:
  1. Add the reliable messaging policy to your copy of the secure conversation policy set.
  2. Specify in the secure conversation bindings that the reliable messaging headers must be signed. For more information about how to do this, see “Defining and managing policy set bindings” on page 824.
  3. Save your changes to the master configuration.

  Note: The reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND) are specifically designed and configured to use secure conversation and reliable messaging in the same policy set. Only create your policy set from a copy of the secure conversation policy set if you have a clear business need to do so. Otherwise, always use a copy of one of the reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND).

Configure secure conversation to increase the timeout setting for the scoping security context token:

About this task

When you use a persistent WS-I RSP policy set, which includes WS-SecureConversation, if the scoping security context token is expired when the server is restarted then WS-ReliableMessaging cannot resend its messages and system messages are written to the log file stating that the reliable messaging sequence was not secured using the correct security token.

To ensure that the scoping security context token does not expire before WS-ReliableMessaging can recover and resend its messages, use the administrative console to complete the following steps:

1. In the navigation pane, click **Services** → **Security cache**. The Security cache detail form is displayed.
2. Set the **Time token is in cache after timeout** property to a value of at least 120 minutes. This value specifies the length of time to keep tokens after they expire. By default, this is 10 minutes. These expired tokens can be used for reliable messaging recovery.
3. In the navigation pane, click **Services** → **Trust service** → **Token providers**. In the content pane, select a security context token. The Security Context Token detail form is displayed.
4. Set the following values for the security context token:
   a. Check that the **Time in cache after expiration** property is set to a value of at least 120 minutes.
   b. Check that the **Token timeout** property is set to a value of at least 120 minutes.
   c. Select the **Allow renewal after timeout** check box.
5. Save your changes to the master configuration.

Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. Use the administrative console to attach the policy set to your application, and (for managed qualities of service) define bindings to a service integration bus and messaging engine.

Before you begin

You can attach a WS-ReliableMessaging policy set and define bindings using the administrative console as described in this task, or you can attach and bind a WS-ReliableMessaging policy set using the wsadmin tool.

This task assumes that you have already developed and installed the Web service application to which you want to attach a policy set.

The following default policy sets work with WS-ReliableMessaging applications:

- WS-I RSP
- WS-I RSP ND
- LTPA WS-I RSP
- Username WS-I RSP
- WSReliableMessaging 1_0
- WSReliableMessaging default
- WSReliableMessaging persistent

For more information, see “WS-ReliableMessaging default policy sets” on page 811.

If you can use any of these default policy sets, or you have configured your own reliable messaging policy set, then you are ready to complete this task.

About this task

Use this task to complete the following broad actions:

1. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or Web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
2. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

To attach a WS-ReliableMessaging policy set and define bindings using the administrative console, complete the following steps:

1. Attach a policy set to your reliable messaging application at either application level or service level..

   Note:

   - You can attach one policy set at each level.
   - You can only apply a WS-ReliableMessaging policy at application level or service level.
   - If you apply reliable messaging at service level, then all services must use the same WS-ReliableMessaging policy and bindings values.
   - You can attach any policy set at operation level. For a policy set that includes the WS-ReliableMessaging policy, attachment at the operation level configures the other components of the policy set (for example WS-Security and WS-Addressing) but any WS-ReliableMessaging configuration at operation level is ignored.
2. If your chosen policy set specifies a managed quality of service, define bindings to a service integration bus and messaging engine.

   If the policy set instance specifies managed non-persistent or managed persistent quality of service, choose the service integration bus and messaging engine that is to manage the

WS-ReliableMessaging state. Use the “WS-ReliableMessaging policy binding” on page 910 panel to select or create the service integration bus and messaging engine that you want to use.

Note: When many applications use the same messaging engine, it can impact performance. Factors to consider include the number of applications that are already binding to the messaging engine, the CPU utilization, and the message throughput. To improve performance for a single server configuration, create a new messaging engine to bind to your application.

To define the default WS-ReliableMessaging policy binding for provider and client policy set attachments within WebSphere Application Server Version 6.1 applications, and for attachments to service applications that are deployed to a Version 6.1 server, navigate to **Services** → **Policy sets** → **Default policy set bindings** → **Version 6.1 default policy set bindings** → **WS-ReliableMessaging**.

To define the bindings for a WebSphere Application Server Version 7.0 provider or client policy set, navigate to **Services** → **Policy sets** → **General provider policy set bindings** → ***provider_policy_set_binding_name*** → **WS-ReliableMessaging** or **Services** → **Policy sets** → **General client policy set bindings** → ***client_policy_set_binding_name*** → **WS-ReliableMessaging**

To define the bindings for an application that you have attached to a service provider policy set, navigate to **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **[Web Service Properties] Service provider policy sets and bindings** and follow the instructions given in “Managing policy sets and bindings for service providers at the application level using the administrative console” on page 743.

To define the bindings for an application that you have attached to a service client policy set, navigate to **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **[Web Service Properties] Service client policy sets and bindings** and follow the instructions given in “Managing policy sets and bindings for service clients at the application level using the administrative console” on page 760.

Note: If the application that you have attached to a service client policy set is a Version 7.0 WS-Notification service client, you can instead use the context-specific version of the “Service client policy sets and bindings” panel that can be reached through either of the following paths:

- **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**
- **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**

If you want to configure policy set and binding details for a single Version 7.0 WS-Notification service client, rather than for all clients for the service, you can instead use the following panel:

- **Services** → **Service clients** → ***ws-notification_service_client_name***

This panel also gives you links to the associated service integration bus and WS-Notification service.

3. Save your changes to the master configuration.

Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the wsadmin tool

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. Use the wsadmin tool to attach the policy set to your application, and (for managed qualities of service) to define bindings to a service integration bus and messaging engine.

Before you begin

You can attach a WS-ReliableMessaging policy set and define bindings using the wsadmin tool as described in this task, or you can attach and bind a WS-ReliableMessaging policy set using the administrative console.

This task assumes that you have already developed and installed the Web service application to which you want to attach a policy set.

The following default policy sets work with WS-ReliableMessaging applications:

- WS-I RSP
- WS-I RSP ND
- LTPA WS-I RSP
- Username WS-I RSP
- WSReliableMessaging 1_0
- WSReliableMessaging default
- WSReliableMessaging persistent

For more information, see “WS-ReliableMessaging default policy sets” on page 811.

If you can use any of these default policy sets, or you have configured your own reliable messaging policy set, then you are ready to complete this task.

About this task

Use this task to complete the following broad actions:

1. Attach a reliable messaging policy set (either a default policy set or one that you have created) to an aspect of your application (that is, application level or Web service level). Policy sets define the reliability level (quality of service) and other configuration options that you want to apply to your reliable messaging application.
2. Define the bindings for each attachment to a policy set that specifies a managed quality of service. That is, choose the service integration bus and messaging engine to use to maintain the state for the managed persistent and managed non-persistent qualities of service.

To attach a WS-ReliableMessaging policy set and define bindings using the wsadmin tool, complete the following steps:

1. Attach a policy set to your reliable messaging application at either application level or service level.

   Use the createPolicySetAttachment command as described in Creating policy set attachments using the wsadmin tool. Set the **-policySet** parameter to the name of the reliable messaging policy set that you want to use. For example: “WS-I RSP ND”

   Note:

   - You can attach one policy set at each level.
   - You can only apply a WS-ReliableMessaging policy at application level or service level.
   - If you apply reliable messaging at service level, then all services must use the same WS-ReliableMessaging policy and bindings values.
   - You can attach any policy set at operation level. For a policy set that includes the WS-ReliableMessaging policy, attachment at the operation level configures the other components of the policy set (for example WS-Security and WS-Addressing) but any WS-ReliableMessaging configuration at operation level is ignored.

   This command returns an attachment ID number. If your chosen policy set specifies a managed quality of service, make a note of this number. In the next step you use it to define the binding.
2. If your chosen policy set specifies a managed quality of service, define bindings to a service integration bus and messaging engine.

   If the policy set instance specifies managed non-persistent or managed persistent quality of service, choose the service integration bus and messaging engine that is to manage the WS-ReliableMessaging state. Use the setBinding command as described in Creating policy set

attachments using the wsadmin tool. Set the **-policyType** parameter to WSReliableMessaging. Set the bus name and the messaging engine name by using the following syntax for the **-attributes** parameter:

```
-attributes "[[busName ReliableMessagingBus] [messagingEngineName nodeName.serverName-busName]]"
```

Note: When many applications use the same messaging engine, it can impact performance. Factors to consider include the number of applications that are already binding to the messaging engine, the CPU utilization, and the message throughput. To improve performance for a single server configuration, create a new messaging engine to bind to your application.

3. Save your changes to the master configuration.

   To save your configuration changes, enter the following command:

   ```
   AdminConfig.save()
   ```

Detecting and fixing problems with WS-ReliableMessaging

The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

About this task

If a sequence fails, a message is written to the application server `SystemOut.log` file and a system event is generated. Therefore you can detect failed sequences by looking at the `SystemOut.log` file, or by writing an event listener (or using third party software) to monitor system events.

Note: After a sequence has been established, WS-ReliableMessaging provides retransmission of messages to a service. However if the sequence is not established (that is, if the initial **CreateSequence** request is refused) then the messages are not transmitted to the service. For more information, see the troubleshooting tip “A sequence is not established, and therefore WS-ReliableMessaging cannot ensure messages are transmitted” on page 667.

For more detailed status information at run time, and facilities to help you fix problems, use the WS-ReliableMessaging administrative console runtime panels. These panels are available at many different scopes (for example cell; application server; messaging engine). For a full list of the WS-ReliableMessaging runtime panels, and details of the scopes at which they are available, see “WS-ReliableMessaging - administrative console panels” on page 671.

At all scopes, the parent panel is “Reliable messaging state settings” on page 672. From this panel you can investigate each of the three key runtime aspects of reliable messaging:

- Message stores
- Inbound sequences
- Outbound sequences

The following icons are displayed here and on several other reliable messaging runtime panels:

	**OK**	Everything here, and (if there is a link) in all runtime panels below this link, is running normally.

	**Warning**	Something here, or (if there is a link) in one of the runtime panels below this link, is in an unusual state and you might need to take some action to resolve it. For example, the system might be awaiting a response from an endpoint. In this case, either the response will be received (in which case you need take no action and the runtime information will be updated to ″OK″) or the reliable messaging destination has stopped acknowledging messages (in which case you need to take some action to resolve the failed sequence).
	**Error**	There is a definite error that you need to take some action to resolve, either here or (if there is a link) in one of the runtime panels below this link.

Note that for troubleshooting purposes you only need to follow links to the sub-panels if states other than ″OK″ are displayed.

To use the reliable messaging runtime panels to detect and fix problems with WS-ReliableMessaging, complete one or more of the following steps:

- Investigate problems with message stores.

  In the navigation pane, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Message store**. The list of reliable messaging storage managers for the current scope is displayed in the “Message store collection” on page 673 form.

  For the managed qualities of service, the messages are written to a messaging engine. For the unmanaged non-persistent quality of service, the messages are stored in memory. For in-memory stores the only possible value is ″Running″. For messages stored by a messaging engine, the possible values are ″Running″ or ″Messaging engine not contactable″, probably because the messaging engine is not running. The ″OK″ icon indicates that the message store is running. If the messaging engine is not contactable, the ″Error″ icon is displayed.

  For each message store in the list, the name of the associated reliable messaging application is given in the *description* column. If a messaging engine is not contactable, restart the message store for that application.

- Investigate problems with inbound sequences.

  In the navigation pane, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences**. The runtime state of each of the inbound sequences for the current scope is displayed in the “Inbound sequence collection” on page 679 form.

  You can use a filter to look at sequences that are in a particular state (for example “Failed due to missing message”) or that have a large number of messages awaiting dispatch to applications. If the sequence status is *Error*, there is a problem with the sequence and the source server hosting the other end of the sequence has terminated it. If the sequence is active and there are a large number of messages awaiting dispatch to the application, then there could be a problem with the application or, if in-order delivery is specified, delivery could be held up because the sequence has gaps in it.

  You can select one or more sequences, then use the buttons provided to dispatch the messages to their associated applications, to export the messages to ZIP files, to close or terminate the selected sequences, or to delete the selected sequences and all their messages.

  To see more detailed information about a particular sequence, click the Sequence identifier field. The “Inbound sequences settings” on page 681 form is displayed. This detailed information includes addressing information to help you identify the source of the sequence, and the value (true or false) for ″in-order delivery″ for the sequence. From this panel you can also display the following forms:

  – The “Acknowledgement state collection” on page 683 form. (The ranges of message sequence numbers received from the WS-ReliableMessaging source. If more than one range is displayed, this indicates a gap in the messages received. If ″In-order delivery″ is selected for the sequence manager, messages with a sequence number greater than the lowest gap cannot be delivered to the application until the gap is closed.)

  - The “Inbound message collection” on page 683 form. (The messages on the inbound sequence. You can use this form to delete individual messages.)
    - The “Message settings” on page 678 form. (The contents of an individual message in the sequence.)

  For more guidance on diagnosing problems with inbound sequences, see “Diagnosing the problem when a reliable messaging source cannot deliver its messages”

- Investigate problems with outbound sequences.

  In the navigation pane, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences**. The runtime state of each of the outbound sequences for the current scope is displayed in the “Outbound sequence collection” on page 674 form.

  You can use a filter to look at sequences that are in a particular state. For example, the state “Cannot contact the remote endpoint” indicates that the sequence has been established but the reliable messaging destination has stopped acknowledging messages (which, coupled with a high number of messages awaiting transmission, could indicate a potential problem). If the sequence status is *Error*, there is a problem with the sequence and the server hosting the other end of the sequence has terminated it.

  You can select one or more sequences, and use one of the buttons provided to reallocate messages to new sequences, to export the messages to ZIP files, to close or terminate the selected sequences, or to delete the selected sequences and all their messages.

  Note: Before you delete a sequence, you can reassign its messages to new sequences or export them to a ZIP file. For more information, see “Deleting a failed WS-ReliableMessaging outbound sequence” on page 664

  To see more detailed information about a particular sequence, click the Sequence identifier field. The “Outbound sequences settings” on page 676 form is displayed. This detailed information includes addressing information to help you identify the server at which the sequence is targeted. From this panel you can also display the following forms:
  - The “Outbound message collection” on page 678 form. (The messages on the outbound sequence. You can use this form to delete individual messages.)
    - The “Message settings” on page 678 form. (The contents of an individual message in the sequence.)

  For more guidance on diagnosing problems with outbound sequences, see “Diagnosing and recovering a WS-ReliableMessaging outbound sequence that is in retransmitting state” on page 664.

Diagnosing the problem when a reliable messaging source cannot deliver its messages

If you know the sequence identifier, and the target URI for the messages, you can use the runtime administrative console panels to examine the sequence and determine why a reliable messaging source is not delivering its messages into the application server.

About this task

To diagnose this problem, complete the following steps:

1. Identify the target server from the target URI.
2. Use the administrative console to view the inbound sequences runtime information for the target server.

   In the navigation pane, click either **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** or **Servers** → **Clusters** → **WebSphere application server clusters** → ***cluster_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences**. The runtime state of each of the inbound sequences for the current scope is displayed in the “Inbound sequence collection” on page 679 form.

3. Specify a filter on the runtime view of the sequences, filtering on the Sequence identifier provided by the reliable messaging source owner.
4. Click on the Sequence identifier of the displayed sequence to get a more detailed view of it.
5. If the sequence has failed, examine the runtime status of the failed sequence and provide the reliable messaging source owner with details of which messages have been acknowledged, so that the owner can decide whether to delete, or to process out of order, any undelivered messages.
6. If the sequence has not failed, see whether there is a communications problem between the reliable messaging source and the application server.

Diagnosing and recovering a WS-ReliableMessaging outbound sequence that is in retransmitting state

You need to resolve a sequence in retransmitting state, because messages are backing up awaiting delivery to the target service. The retransmission could be due to a network failure, or a failure of the target server. The problem should resolve automatically, but you might want to investigate the cause to speed up recovery.

About this task

To resolve a sequence in retransmitting state, complete the following steps:

1. Check for network failures.
2. If no network failures are found, look up the target endpoint address (URI) for the sequence to determine the owner of the target service, then contact the owner to check if the target server is available.

   The target endpoint address is shown in the “Outbound sequences settings” on page 676 form. Use the administrative console to navigate to this form, as described in “Detecting and fixing problems with WS-ReliableMessaging” on page 661.
3. After the communication link between the two servers is re-established, use the runtime panels to confirm that messages in the sequence are now being delivered.

Deleting a failed WS-ReliableMessaging outbound sequence

You need to resolve an outbound sequence in failed state, so that messages can again be transmitted to the target service. A sequence in failed state shows an unrecoverable error. The sequence can no longer be used. If messages are being delivered in order, then the failed sequence must be resolved before a new sequence can be established.

About this task

Deleting an outbound sequence allows the runtime environment to automatically create a new sequence the next time that an application attempts to invoke a Web service at the destination address that the failed sequence was targeting. To work with outbound sequences you use the administrative console runtime panels as described in “Detecting and fixing problems with WS-ReliableMessaging” on page 661.

To diagnose and delete a failed outbound sequence, use the administrative console to complete the following steps:

1. In the navigation pane of the administrative console, click one of the paths to the outbound sequences collection form. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences**. The runtime state of each of the outbound sequences for the current scope is displayed in the “Outbound sequence collection” on page 674 form.
2. Examine the failure reason by clicking on the Sequence identifier field of the failed sequence. The “Outbound sequences settings” on page 676 form is displayed. The failure reason is based on the fault message received by the sequence manager from the target server.
3. If there are messages associated with the failed sequence, decide what to do with these messages. The messages might have been transmitted and received at the target server, or they might not. You

might choose to delete messages from the sequence, re-allocate them to a new sequence or export them to a ZIP file. If you choose to delete the messages, you can either delete individual messages or you can delete all the messages.

a. Optional: To delete one or more messages from a failed sequence, complete the following steps:
   1) In the main pane of the "Outbound sequences settings" on page 676 form, under the Additional Properties section, click **Messages**. The messages for the failed outbound sequence are listed in the "Outbound message collection" on page 678 form.
   2) Select the check boxes next to the names of the messages that you want to delete.
   3) Click **Delete**.

b. Optional: To re-allocate or export all the remaining messages in a failed sequence, complete the following steps:
   1) In the main pane of the "Outbound sequence collection" on page 674 form, select the check box next to the name of the failed sequence.
   2) Click **Re-allocate messages** or **Export unsent messages**. All remaining messages in the sequence are re-allocated to a new sequence or exported to a ZIP file.

4. Close or terminate the failed sequence.

   Note: In the WS-ReliableMessaging Version 1.1 specification, a sequence can be closed rather than terminated. This allows the final ACK state to be sent from the reliable messaging destination to the reliable messaging source. In the WS-ReliableMessaging Version 1.0 specification this does not happen, so the final ACK state might not be known at the reliable messaging source. For more information about the distinction between close and terminate, see "Outbound sequence collection" on page 674.

   a. In the main pane of the "Outbound sequence collection" on page 674 form, select the check box next to the name of the failed sequence.
   b. Click **Close sequence** or **Terminate sequence**.

5. Delete the failed sequence.
   a. In the main pane of the "Outbound sequence collection" on page 674 form, select the check box next to the name of the failed sequence.
   b. Click **Delete sequence**.

WS-ReliableMessaging troubleshooting tips

Tips for troubleshooting your WS-ReliableMessaging configuration.

To help you identify and resolve problems with WS-ReliableMessaging, you can use the TCP/IP monitor to view the messages that are flowing between your client applications and reliable messaging enabled Web services. You can also use the WebSphere Application Server trace and logging facilities as described in Tracing and logging configuration.

To enable trace for WS-ReliableMessaging, set the application server trace string as follows:

- For either of the managed qualities of service:

```
org.apache.sandesha2*=all=enabled:com.ibm.ws.websvcs.rm*=all=enabled:org.apache.axis2*=all=enabled:com.ibm.ws.sib.wsrm*=all=enabled
```

- For the Unmanaged Non-Persistent quality of service:

```
org.apache.sandesha2*=all=enabled:com.ibm.ws.websvcs.rm*=all=enabled:org.apache.axis2*=all=enabled
```

If you encounter a problem that you think might be related to WS-ReliableMessaging, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `SystemOut.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the main known restrictions that apply when using WS-ReliableMessaging is provided in "WS-ReliableMessaging known restrictions" on page 669.

WebSphere Application Server system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message. The prefix for the WS-ReliableMessaging component is CWSKA.

The Troubleshooter reference: Messages topic contains information about all WebSphere Application Server messages, indexed by message prefix. For each message there is an explanation of the problem, and details of any action that you can take to resolve the problem.

Here is a set of tips to help you troubleshoot commonly-experienced problems:
- "If reliable messaging is running on a cluster, when you examine the runtime state of inbound or outbound sequences you see multiple entries for each sequence"
- "You get runtime errors when you migrate persisted WS-ReliableMessaging messages from Version 6.1.0.9 or 6.1.0.11 of the Feature Pack for Web Services to a later version of the product"
- "When an application server starts, a messaging engine used for reliable messaging is reported as unavailable" on page 667
- "A client application is unable to invoke a reliable messaging enabled Web service" on page 667
- "A sequence is not established, and therefore WS-ReliableMessaging cannot ensure messages are transmitted" on page 667
- "A sequence is established but cannot be used, and therefore WS-ReliableMessaging cannot ensure messages are transmitted" on page 668
- "The reliable messaging managed store is not initialized because the policy set binding is not complete or not valid" on page 668
- "A message is not recovered after a server becomes unavailable, even with the managed persistent quality of service" on page 668
- "When using reliable messaging with a persistent WS-I RSP profile and WS-SecureConversation, an exception message states that the security context token is not valid" on page 669

If reliable messaging is running on a cluster, when you examine the runtime state of inbound or outbound sequences you see multiple entries for each sequence

This is because, although reliable messaging only binds to one messaging engine in a cluster, the runtime panel is calculating and displaying the sequence information once for every cluster member. Ignore the duplicate entries. Note that the slight differences in the statistics being displayed for each duplicate entry is due to the entries being created sequentially, while polling for messages continues.

You get runtime errors when you migrate persisted WS-ReliableMessaging messages from Version 6.1.0.9 or 6.1.0.11 of the Feature Pack for Web Services to a later version of the product

If you are migrating from WebSphere Application Server Version 6.1, and you are using Version 6.1.0.9 or 6.1.0.11 of the Feature Pack for Web Services, and your configuration includes WS-ReliableMessaging configured for the managed persistent quality of service, you need to remove all persisted messages before you migrate.

Each message is persisted as part of a sequence that is currently being processed. To remove all persisted messages, use the administrative console to complete the following steps:

1. Navigate to the "Inbound sequence collection" on page 679 runtime panel for your reliable messaging application.
2. Select all the inbound sequences, then click the **delete sequence and messages** button to delete the sequences.
3. Navigate to the "Outbound sequence collection" on page 674 runtime panel, then repeat the previous steps for the outbound sequences.

When an application server starts, a messaging engine used for reliable messaging is reported as unavailable

When you use reliable messaging with a managed quality of service, you might see the following exception message when your application server starts:

```
CWSIT0019E: No suitable messaging engine is available on bus yourBus that matched the specified connection properties
```

If you suspect there is an underlying problem, for example the bindings are incorrect or the server that hosts the messaging engine is not going to start, complete the following checks:

- Check that the specified messaging engine and service integration bus exist.
- Check the system out log to ensure that the server that hosts the messaging engine has started.

A client application is unable to invoke a reliable messaging enabled Web service

If your client application is unable to invoke a reliable messaging enabled Web service, you can use the TCP-IP monitor to view the messages that are flowing between the client and the service. You should also check the following:

- The endpoint is available.
- The service is running.
- The service has been invoked.
- WS-ReliableMessaging is running.
- WS-ReliableMessaging is correctly configured. In particular, for either of the managed qualities of service, check that you have configured a valid binding to a service integration bus and messaging engine, and that the messaging engine is running. For more information, see “Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console” on page 657 or “Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the wsadmin tool” on page 659.
- There are not too many applications sharing a single messaging engine.

  Note: When many applications use the same messaging engine, it can impact performance. Factors to consider include the number of applications that are already binding to the messaging engine, the CPU utilization, and the message throughput. To improve performance for a single server configuration, create a new messaging engine to bind to your application.

A sequence is not established, and therefore WS-ReliableMessaging cannot ensure messages are transmitted

After a sequence has been established, WS-ReliableMessaging provides retransmission of messages to a service. However if the sequence is not established then the messages are not transmitted to the service and a message similar to the following example is displayed:

```
org.apache.axis2.AxisFault: The Create Sequence request has been refused by the RM Destination
```

The initial **createSequence** message has been refused. This is propagated back, and causes the client to fail. For information about **CreateSequence** and **CreateSequenceRefused**, see the “WS-ReliableMessaging: supported specifications and standards” on page 643.

You might also see a subsequent message to help explain why the request has been refused. For example:

```
Caused by: javax.xml.ws.soap.SOAPFaultException: com.ibm.ws.sib.wsrm.exceptions.WSRMRuntimeException:
CWSJZ0202I: A messaging engine connection is unavailable for bus myBus.
```

There is a problem with your reliable messaging configuration. Complete the following checks:

- Check that the policy sets are correctly applied. Specifically, check that the destination has reliable messaging correctly enabled.
- Check the logs for server-side problems.
- For the managed persistent quality of service, check that the associated messaging engine is available.

For further checks that might help you resolve the problem, see also the following troubleshooting tips:
- "A sequence is established but cannot be used, and therefore WS-ReliableMessaging cannot ensure messages are transmitted."
- "A client application is unable to invoke a reliable messaging enabled Web service" on page 667.

A sequence is established but cannot be used, and therefore WS-ReliableMessaging cannot ensure messages are transmitted

If you get an exception like the following exception, then the sequence is established but cannot be used:

```
javax.xml.ws.WebServiceException: org.apache.axis2.AxisFault: The value of wsrm:Identifier is not a known Sequence identifier.
```

For checks that might help you resolve the problem, see the following troubleshooting tips:
- "A sequence is not established, and therefore WS-ReliableMessaging cannot ensure messages are transmitted" on page 667.
- "A client application is unable to invoke a reliable messaging enabled Web service" on page 667.

The reliable messaging managed store is not initialized because the policy set binding is not complete or not valid

If your policy set specifies a managed quality of service, but you have not specified a binding to a messaging engine to support that quality of service, you get the following exception message:

```
CWSKA0102E: The managed Web services reliable messaging storage manager could not be initialized because the policy set binding
 was incomplete or invalid.
```

Perhaps you have attached a managed policy set to your application, and used the default bindings (which do not support the managed qualities of service). You need to create a new binding for your application that specifies a service integration bus and messaging engine to support the managed qualities of service. To do this, see "Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console" on page 657.

A message is not recovered after a server becomes unavailable, even with the managed persistent quality of service

When the reliable messaging layer receives a request message, it sends an acknowledgement then delivers the message to the target service. There is a marginal possibility that the server hosting the reliable messaging layer could become unavailable after the request message has been acknowledged and before it has been delivered. In this case, the message is only recovered if you are using in-order delivery as well as managed persistent quality of service. To specify in-order delivery, select the WS-ReliableMessaging policy option to "Deliver messages in the order that they were sent" as described in "Configuring the WS-ReliableMessaging policy" on page 907.

Note: There is a performance overhead in using in-order delivery, because messages are held in a queue until they can be delivered in order. However, where the highest level of reliability is paramount it is recommended that you always specify in-order delivery in conjunction with the managed persistent quality of service.

When using reliable messaging with a persistent WS-I RSP profile and WS-SecureConversation, an exception message states that the security context token is not valid

When you use a persistent WS-I RSP policy set, which includes WS-SecureConversation, if the scoping security context token is expired when the server is restarted then WS-ReliableMessaging cannot resend its messages and system messages are written to the log file stating that the reliable messaging sequence was not secured using the correct security token. For example:

```
CWWSS7215E: Cannot get valid security context token from the cache.
```

To ensure that the scoping security context token does not expire before WS-ReliableMessaging can recover and resend its messages, complete the following task: “Configuring WS-SecureConversation to work with WS-ReliableMessaging” on page 656.

WS-ReliableMessaging known restrictions:

The main known restrictions that apply to the implementation of WS-ReliableMessaging in WebSphere Application Server.

- “IBM HTTP Server is not supported as a proxy server with WS-ReliableMessaging”
- “A WS-ReliableMessaging policy can only be applied at application or service level.”
- “WS-SecureConversation must expect the WS-ReliableMessaging headers to be signed”

IBM HTTP Server is not supported as a proxy server with WS-ReliableMessaging

When WS-ReliableMessaging is enabled, you cannot use IBM HTTP Server as a proxy server to route responses back to an individual server in a clustered environment. If you use IBM HTTP Server in this scenario, replies to protocol messages, such as WS-ReliableMessaging CreateSequenceResponse messages, are not routed back to the originating server. This is because WS-ReliableMessaging protocol messages use WS-Addressing affinity headers to ensure that the protocol responses are routed back to the originating server. These affinity headers are supported in IBM WebSphere Application Server proxy server, but are not supported in IBM HTTP Server.

Use the IBM WebSphere Application Server proxy server to route messages from and back to the cluster, instead of IBM HTTP Server.

A WS-ReliableMessaging policy can only be applied at application or service level.

You can only apply a WS-ReliableMessaging policy at application level or service level.

If you apply reliable messaging at service level, then all services must use the same WS-ReliableMessaging policy and bindings values.

You can attach any policy set at operation level. For a policy set that includes the WS-ReliableMessaging policy, attachment at the operation level configures the other components of the policy set (for example WS-Security and WS-Addressing) but any WS-ReliableMessaging configuration at operation level is ignored.

WS-SecureConversation must expect the WS-ReliableMessaging headers to be signed

Although secure conversation allows message headers to remain unsigned, the reliable messaging policy requires the reliable messaging headers to be signed. If you want to use secure conversation and reliable messaging policies in the same policy set, the secure conversation bindings must be configured to require that the reliable messaging headers are signed.

The reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND) are specifically designed and configured to use secure conversation and reliable messaging in the same policy set. If you use a copy of

one of the reliable secure profile default policy sets (WS-I RSP and WS-I RSP ND), no further configuration of the secure conversation bindings is required. Otherwise, see “Configuring WS-SecureConversation to work with WS-ReliableMessaging” on page 656.

WS-ReliableMessaging roles and goals

Computing roles that members of your organization might perform, and how you can use WS-ReliableMessaging to help meet the goals of each role.

For a general description of each of the following roles, see “WebSphere Application Server roles and goals” on page 397.

Application developer

The application developer is responsible for creating the WS-ReliableMessaging requester applications and provider applications. The application developer is responsible for writing the application code and packaging the application into a deployable unit.

Developing

- Goal: Develop a JAX-WS Web service application.

System administrator

The system administrator is responsible for providing the infrastructure, such as configured application servers, through which the applications can be deployed and executed. The system administrator is also responsible for deploying applications into the configured environment, and for maintaining the environment and applications in good working order. This maintenance role includes operational management of state, such as messages and transaction state, and problem diagnosis to determine message or transaction outcomes.

Deploying

- Goal: Configure a policy set instance to enable WS-ReliableMessaging.
- Goal: Install your reliable JAX-WS Web service application.
- Goal: Attach and bind a WS-ReliableMessaging policy set to your application.

Operating

- Goal: Detect and fix problems with WS-ReliableMessaging.

WS-ReliableMessaging - requirements for interaction with other implementations

The information and configuration that is needed for another vendor’s reliable messaging source to send messages to a WebSphere Application Server reliable messaging destination, or for a WebSphere Application Server reliable messaging source to send messages to another vendor’s reliable messaging destination.

Using another vendor’s reliable messaging source to send messages to a WebSphere Application Server reliable messaging destination

For another vendor’s WS-ReliableMessaging implementation to interact with a WebSphere Application Server WS-ReliableMessaging endpoint, the other vendor’s reliable messaging source needs to know the target endpoint address and port for the WebSphere Application Server application that is enabled for reliable messaging. The WS-ReliableMessaging protocol messages are sent to the same endpoint address as the application messages. You can get this address from the WSDL published by the WebSphere Application Server endpoint.

The reliable messaging source controls the endpoint reference used for acknowledgement messages, so the other vendor's product might need to use the WS-Addressing anonymous URI. For more information, see WS-ReliableMessaging - How it works. Whether or not the reliable messaging source uses the WS-Addressing anonymous URI, the WebSphere Application Server reliable messaging destination can work with the reliable messaging source without further configuration.

A WebSphere Application Server reliable messaging destination cannot tell whether or not the reliable messaging source is durable and transactional. If you want durable transactional Web services, check that the other vendor's reliable messaging source supports that mode of operation, as well as configuring the WebSphere Application Server end of the link.

Using a WebSphere Application Server reliable messaging source to send messages to another vendor's reliable messaging destination

For an application in WebSphere Application Server to invoke a Web service using WS-ReliableMessaging, the information required is the target endpoint address and port for the Web service being invoked. The WS-ReliableMessaging protocol messages are sent to the same endpoint address as the application messages. You can usually get this address from the WSDL published by the target Web service.

The WebSphere Application Server Source is provided with additional WS-ReliableMessaging configuration, modeled as part of the policy set associated with the Web service client. The policy set might configure the reliable messaging source to use the WS-Addressing anonymous URI as the address within the endpoint reference used for acknowledgement messages. For more information, see "WS-ReliableMessaging - How it works" on page 636

WebSphere Application Server cannot tell whether the reliable messaging destination is durable and transactional. If you want durable transactional Web services, check that the other vendor's reliable messaging destination supports that mode of operation, as well as configuring the WebSphere Application Server end of the link.

WS-ReliableMessaging - administrative console panels

Links to topics that describe the contents of the administrative console panels that you can use to configure and operate WS-ReliableMessaging. Each topic gives details of the purpose and use of a panel, the administrative console navigation path to the panel, and the values that you can set in each field of the panel.

Configuration panels

- "WS-ReliableMessaging settings" on page 908
- "WS-ReliableMessaging policy binding" on page 910

Runtime panels

This set of panels is available at any of the following scopes within the administrative console:

Cell: **Services** → **Reliable messaging state**

Application server:
Servers → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state**

Enterprise application:
Applications → **Enterprise Applications** → ***application_name*** → **[Web Services Properties] Reliable messaging state**

Bus: **Service integration** → **Buses** → ***bus_name*** → **[Services] Reliable messaging state**

Messaging engine:
Service integration → Buses → *bus_name* → [Topology] Messaging engines → *messaging_engine_name* → [Additional Properties] Reliable messaging state

At each of the previous scopes, the panels navigation structure is as follows:

- "Reliable messaging state settings"
  - "Message store collection" on page 673
  - "Inbound sequence collection" on page 679
    - "Inbound sequences settings" on page 681
      - "Acknowledgement state collection" on page 683
      - "Inbound message collection" on page 683
        - "Message settings" on page 678
    - "Export messages settings" on page 684
  - "Outbound sequence collection" on page 674
    - "Outbound sequences settings" on page 676
      - "Outbound message collection" on page 678
        - "Message settings" on page 678
    - "Export messages settings" on page 684

Reliable messaging state settings

This page provides an overview of the WS-ReliableMessaging runtime state. Use this page to manage reliable messaging at run time.

To view this pane in the console, click one of the paths to this panel. For example **Servers → Server Types → WebSphere application servers → *server_name* → [Additional Properties] Reliable messaging state**.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see "WS-ReliableMessaging - administrative console panels" on page 671.

Related tasks

"Detecting and fixing problems with WS-ReliableMessaging" on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Runtime tab: Runtime properties for this object. These properties directly affect the current runtime environment, but are not preserved when that environment is stopped. To preserve runtime property values, change the equivalent property values on the Configuration tab. See the information center task descriptions for information about how to apply configuration changes to the runtime environment.

Reliable messaging state

Select the aspect of WS-ReliableMessaging for which you want to view runtime information.

The following icons are displayed here and on several other reliable messaging runtime panels:

	**OK**	Everything here, and (if there is a link) in all runtime panels below this link, is running normally.

	**Warning**	Something here, or (if there is a link) in one of the runtime panels below this link, is in an unusual state and you might need to take some action to resolve it. For example, the system might be awaiting a response from an endpoint. In this case, either the response will be received (in which case you need take no action and the runtime information will be updated to ″OK″) or the reliable messaging destination has stopped acknowledging messages (in which case you need to take some action to resolve the failed sequence).
	**Error**	There is a definite error that you need to take some action to resolve, either here or (if there is a link) in one of the runtime panels below this link.

Note that for troubleshooting purposes you only need to follow links to the sub-panels if states other than ″OK″ are displayed.

Message stores
This page displays the collection of reliable messaging storage managers for the current scope.

Inbound sequences
This page displays the collection of inbound sequences for the current scope. Each inbound sequence is used to receive messages that have been transmitted reliably.

Outbound sequences
This page displays the collection of outbound sequences for the current scope. Outbound sequences are used to transmit messages reliably from the local application to the remote endpoint. Each sequence has a unique identifier.

Message store collection

This page displays the collection of reliable messaging storage managers for the current scope.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Message store**.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

The following icons are displayed here and on several other reliable messaging runtime panels:

	**OK**	Everything here, and (if there is a link) in all runtime panels below this link, is running normally.
	**Warning**	Something here, or (if there is a link) in one of the runtime panels below this link, is in an unusual state and you might need to take some action to resolve it. For example, the system might be awaiting a response from an endpoint. In this case, either the response will be received (in which case you need take no action and the runtime information will be updated to ″OK″) or the reliable messaging destination has stopped acknowledging messages (in which case you need to take some action to resolve the failed sequence).
	**Error**	There is a definite error that you need to take some action to resolve, either here or (if there is a link) in one of the runtime panels below this link.

Note that for troubleshooting purposes you only need to follow links to the sub-panels if states other than ″OK″ are displayed.

Message store type
For the managed qualities of service, the messages are written to a messaging engine. For the unmanaged non-persistent quality of service, the messages are stored in memory.

Description
The quality of service being used and the application name.

Details
For in-memory stores the only possible value is ″Running″. For messages stored by a messaging engine, the possible values are ″Running″ or ″Messaging engine not contactable″, probably because the messaging engine is not running.

Status
The ″OK″ icon indicates that the message store is running. If the messaging engine is not contactable, the ″Error″ icon is displayed.

Related tasks

“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Outbound sequence collection

This page displays the collection of outbound sequences for the current scope. Outbound sequences are used to transmit messages reliably from the local application to the remote endpoint. Each sequence has a unique identifier.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

The following icons are displayed here and on several other reliable messaging runtime panels:

	**OK**	Everything here, and (if there is a link) in all runtime panels below this link, is running normally.
	**Warning**	Something here, or (if there is a link) in one of the runtime panels below this link, is in an unusual state and you might need to take some action to resolve it. For example, the system might be awaiting a response from an endpoint. In this case, either the response will be received (in which case you need take no action and the runtime information will be updated to ″OK″) or the reliable messaging destination has stopped acknowledging messages (in which case you need to take some action to resolve the failed sequence).
	**Error**	There is a definite error that you need to take some action to resolve, either here or (if there is a link) in one of the runtime panels below this link.

Note that for troubleshooting purposes you only need to follow links to the sub-panels if states other than ″OK″ are displayed.

Note:

If reliable messaging is running on a cluster, when you examine the runtime state of inbound or outbound sequences you see multiple entries for each sequence. This is because, although reliable messaging only binds to one messaging engine in a cluster, the runtime panel is calculating and displaying the sequence information once for every cluster member.

Ignore the duplicate entries. Note that the slight differences in the statistics being displayed for each duplicate entry is due to the entries being created sequentially, while polling for messages continues.

Sequence identifier
This URI is the unique identifier for the sequence.

Associated application
The application that created the sequence.

Target endpoint URI
The destination to which messages are transmitted.

Message depth
The count of messages, held by the reliable messaging layer, that have not yet been transferred.

Messages sent
The total count of messages sent by the application for this sequence.

Details
The current state of the sequence. ″Establishing″ indicates that the sequence is awaiting a CreateSequenceResponse message from the reliable messaging destination; ″Active″ indicates that the sequence has been established; ″Cannot contact the remote endpoint″ indicates that the sequence has been established but the reliable messaging destination has stopped acknowledging messages; ″Sequence closing″ indicates that the sequence is closing and will accept no new messages; ″Sequence closed″ indicates that the sequence has been closed and will accept no new messages; ″Sequence terminating″ indicates that the sequence is terminating and will accept no new messages; ″The inbound side has lost state in a terminate″ indicates that the sequence has terminated with unacknowledged messages outstanding; ″The sequence has timed out″ indicates that the sequence has timed out.

Status
The icon indicates ″OK″, ″Warning″ or ″Error″ as previously described on this page. A more precise indication of the sequence state is given in the ″Details″ column.

Buttons

Re-allocate messages	Click this button to re-allocate the messages on the selected sequences to new sequences.
Export unsent messages	Click this button to export the messages from the selected sequences to ZIP files. If you select any sequences that have no messages to export, a warning or error message is displayed.

Close sequence	Click this button to close the selected sequences and send a CloseSequence message to indicate that the sequence is complete and will not be sending any further messages. The *close* protocol does not delete the resources for the sequence, so you can still access any undispatched or unsent messages. If you are using Version 1.0 of the WS-ReliableMessaging specification (which does not support this option) an error message is displayed. For more information on the reliable messaging *close* protocol, see “WS-ReliableMessaging: supported specifications and standards” on page 643.
Terminate sequence	Click this button to close the selected sequences and send a TerminateSequence message to indicate that the sequence is complete and will not be sending any further messages. The *terminate* protocol deletes all resources for the sequence. For more information on the reliable messaging *terminate* protocol, see “WS-ReliableMessaging: supported specifications and standards” on page 643.
Delete sequence	Click this button to delete the selected sequences without sending a TerminateSequence message.

Related tasks

“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Outbound sequences settings

This page displays the collection of outbound sequences for the current scope. Outbound sequences are used to transmit messages reliably from the local application to the remote endpoint. Each sequence has a unique identifier.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences** → ***outbound_sequence_name***.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

Related tasks

“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Runtime tab: Runtime properties for this object. These properties directly affect the current runtime environment, but are not preserved when that environment is stopped. To preserve runtime property values, change the equivalent property values on the Configuration tab. See the information center task descriptions for information about how to apply configuration changes to the runtime environment.

General properties

Sequence identifier
This URI is the unique identifier for the sequence.

Required	No
Data type	Text

Runtime identifier
The identifier that is used within the application server runtime environment.

Required	No
Data type	Text

Associated application
The application that created the sequence.

Required	No
Data type	Text

WS-Addressing namespace
The WS-Addressing namespace that is associated with the sequence.

Required	No
Data type	Text

WS-ReliableMessaging namespace
The namespace that is defined by the version of the WS-ReliableMessaging specification used by the sequence.

Required	No
Data type	Text

Target endpoint URI
The destination to which messages are transmitted.

Required	No
Data type	Text

Reply to address
The WS-Addressing replyTo address that is used for WS-ReliableMessaging protocol messages.

Required	No
Data type	Text

Acknowledgement address
The address used for WS-ReliableMessaging acknowledgements.

Required	No
Data type	Text

Message depth
The count of messages, held by the reliable messaging layer, that have not yet been transferred.

Required	No
Data type	Text

Messages sent
The total count of messages sent by the application for this sequence.

Required	No
Data type	Text

Outbound message collection
The messages on the outbound sequence.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences** → ***outbound_sequence_name*** → **[Additional properties] Messages**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

Message number
The index number of the message in the sequence.

State The two possible states are ″Sendable″ (indicating that it is expected to be sent soon) and ″Awaiting sequence initialization″.

Buttons

Refresh	Click this button to refresh the list of items.
Delete	Click this button to delete the selected items.

Related tasks

“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Message settings
The details of an individual message.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671. At each scope, this panel is available for both inbound and outbound sequences. For example:

- **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** → ***inbound_sequence_name*** → **[Additional properties] Messages** → ***message_number***

- **Servers → Server Types → WebSphere application servers → *server_name* → [Additional Properties] Reliable messaging state → Runtime → Outbound sequences → *outbound_sequence_name* → [Additional properties] Messages → *message_number***

Related tasks

"Detecting and fixing problems with WS-ReliableMessaging" on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Runtime tab: Runtime properties for this object. These properties directly affect the current runtime environment, but are not preserved when that environment is stopped. To preserve runtime property values, change the equivalent property values on the Configuration tab. See the information center task descriptions for information about how to apply configuration changes to the runtime environment.

General properties

Message number
The index number of the message in the sequence.

Required	No
Data type	Text

Message contents
The contents of the message.

Required	No
Data type	Custom

Inbound sequence collection

This page displays the collection of inbound sequences for the current scope. Each inbound sequence is used to receive messages that have been transmitted reliably.

To view this pane in the console, click one of the paths to this panel. For example **Servers → Server Types → WebSphere application servers → *server_name* → [Additional Properties] Reliable messaging state → Runtime → Inbound sequences**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see "WS-ReliableMessaging - administrative console panels" on page 671.

The following icons are displayed here and on several other reliable messaging runtime panels:

	**OK**	Everything here, and (if there is a link) in all runtime panels below this link, is running normally.

	**Warning**	Something here, or (if there is a link) in one of the runtime panels below this link, is in an unusual state and you might need to take some action to resolve it. For example, the system might be awaiting a response from an endpoint. In this case, either the response will be received (in which case you need take no action and the runtime information will be updated to ″OK″) or the reliable messaging destination has stopped acknowledging messages (in which case you need to take some action to resolve the failed sequence).
	**Error**	There is a definite error that you need to take some action to resolve, either here or (if there is a link) in one of the runtime panels below this link.

Note that for troubleshooting purposes you only need to follow links to the sub-panels if states other than ″OK″ are displayed.

Note:

If reliable messaging is running on a cluster, when you examine the runtime state of inbound or outbound sequences you see multiple entries for each sequence. This is because, although reliable messaging only binds to one messaging engine in a cluster, the runtime panel is calculating and displaying the sequence information once for every cluster member.

Ignore the duplicate entries. Note that the slight differences in the statistics being displayed for each duplicate entry is due to the entries being created sequentially, while polling for messages continues.

Sequence identifier
This URI is the unique identifier for the sequence.

Associated application
The application that is receiving messages from the sequence.

Message depth
The count of messages, held by the reliable messaging layer, that have not yet been transferred.

Messages received
The total count of application messages received for the sequence since its creation.

Details
The current state of the sequence. ″Connected″ indicates that the sequence has been established; ″Awaiting redelivery of a missing msg.″ indicates that there is a gap in the sequence; ″Closed″ indicates that the sequence has been closed; ″Failed due to missing message″ indicates that the sequence terminated with a gap in the sequence.

Status
The icon indicates ″OK″, ″Warning″ or ″Error″ as previously described on this page. A more precise indication of the sequence state is given in the ″Details″ column.

Buttons

Dispatch messages to application	Click this button to dispatch the messages on the selected sequences to the associated applications.
Export undispatched messages	Click this button to export the messages from the selected sequences to ZIP files. If you select any sequences that have no messages to export, a warning or error message is displayed.

Close sequence	Click this button to close the selected sequences and send a CloseSequence message to indicate that the sequence is complete and will not be sending any further messages. The *close* protocol does not delete the resources for the sequence, so you can still access any undispatched or unsent messages. If you are using Version 1.0 of the WS-ReliableMessaging specification (which does not support this option) an error message is displayed. For more information on the reliable messaging *close* protocol, see "WS-ReliableMessaging: supported specifications and standards" on page 643.
Terminate sequence	Click this button to close the selected sequences and send a TerminateSequence message to indicate that the sequence is complete and will not be sending any further messages. The *terminate* protocol deletes all resources for the sequence. For more information on the reliable messaging *terminate* protocol, see "WS-ReliableMessaging: supported specifications and standards" on page 643.
Delete sequence and messages	Click this button to delete the selected sequences and all their messages. Use this in cases where the remote endpoint is unable to respond to any reliable messaging protocol messages, and therefore the sequence cannot be closed or terminated.

Related tasks

"Detecting and fixing problems with WS-ReliableMessaging" on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Inbound sequences settings

This page displays the collection of inbound sequences for the current scope. Each inbound sequence is used to receive messages that have been transmitted reliably.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** → ***inbound_sequence_name***.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see "WS-ReliableMessaging - administrative console panels" on page 671.

Related tasks

"Detecting and fixing problems with WS-ReliableMessaging" on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Runtime tab: Runtime properties for this object. These properties directly affect the current runtime environment, but are not preserved when that environment is stopped. To preserve runtime property values, change the equivalent property values on the Configuration tab. See the information center task descriptions for information about how to apply configuration changes to the runtime environment.

General properties

Sequence identifier
This URI is the unique identifier for the sequence.

Required	No
Data type	Text

Associated application
The application that is receiving messages from the sequence.

Required	No
Data type	Text

WS-Addressing namespace
The WS-Addressing namespace that is associated with the sequence.

Required	No
Data type	Text

WS-ReliableMessaging namespace
The namespace that is defined by the version of the WS-ReliableMessaging specification used by the sequence.

Required	No
Data type	Text

Target endpoint URI
The destination to which messages are transmitted.

Required	No
Data type	Text

Reply to address
The WS-Addressing replyTo address that is used for WS-ReliableMessaging protocol messages.

Required	No
Data type	Text

Acknowledgement address
The address used for WS-ReliableMessaging acknowledgements.

Required	No
Data type	Text

Message depth
The count of messages, held by the reliable messaging layer, that have not yet been transferred.

Required	No
Data type	Text

Highest inbound message number
The highest message number that has been received within the sequence.

Required	No
Data type	Text

In-order delivery
When this parameter is true, messages are dispatched to the application in the order that they were assigned to the sequence.

Required	No
Data type	Text

Inbound message collection
The messages on the inbound sequence.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** → ***inbound_sequence_name*** → **[Additional properties] Messages**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

Message number
The index number of the message in the sequence.

State The only possible state is ″Awaiting dispatch to application″.

Buttons

Refresh	Click this button to refresh the list of items.
Delete	Click this button to delete the selected items.

Related tasks
“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Acknowledgement state collection
The ranges of message sequence numbers received from the WS-ReliableMessaging source. If more than one range is displayed, this indicates a gap in the messages received. If ″In-order delivery″ is selected for the sequence manager, messages with a sequence number greater than the lowest gap cannot be delivered to the application until the gap is closed.

To view this pane in the console, click one of the paths to this panel. For example **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** → ***inbound_sequence_name*** → **[Additional properties] Ack state**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671.

Low end of range
The message number for the lowest numbered message in this range.

High end of range
The message number for the highest numbered message in this range.

Buttons

Refresh	Click this button to refresh the list of items.

Related tasks

“Detecting and fixing problems with WS-ReliableMessaging” on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

Export messages settings

Export the messages from the selected sequences to ZIP files.

The WS-ReliableMessaging runtime panels are available at many different scopes within the administrative console. For more information, see “WS-ReliableMessaging - administrative console panels” on page 671. At each scope, this panel is available for both inbound and outbound sequences. For example:

- **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Inbound sequences** → **[Button bar] Export undispatched messages**
- **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Reliable messaging state** → **Runtime** → **Outbound sequences** → **[Button bar] Export unsent messages**

General properties

Export messages
A list of the exported messages ZIP files that have been created. To save these ZIP files to your file system, click each file individually then use your Web browser ″save file″ option.

Related tasks

"Detecting and fixing problems with WS-ReliableMessaging" on page 661
The nature of WS-ReliableMessaging is that network and server failures are assumed, and therefore the target Web service or message store might not be available. In these cases, message sequences cannot be completed and collections of Web service messages are held awaiting transmission. You can use the `SystemOut.log` file, system events, and the runtime administrative panels to monitor the system and detect and fix problems with WS-ReliableMessaging.

WS-Notification Service client settings

Use this page to manage policy sets and bindings or to access additional information for this WS-Notification service client.

To view this pane in the console, click the following path:

Services → **Service clients** → ***ws-notification_service_client_name***

This panel shows the policy set and binding configuration information for a single Version 7.0 WS-Notification service client, and also gives you links to the associated service integration bus and WS-Notification service.

Note: The same policy set and binding configuration information can also be viewed and modified through the "Service client policy sets and bindings" panel, which gives an upper-level view of all service clients associated with a particular WS-Notification service. Through the "Service client policy sets and bindings" panel, you can assign a policy set or binding to a specific service client, or to all clients for the service. To view the "Service client policy sets and bindings" panel for a given WS-Notification service, click one of the following paths:

- **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**
- **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**

Related concepts

WS-Notification and policy set configuration

"WS-ReliableMessaging default policy sets" on page 811
The WS-ReliableMessaging default policy sets are pre-configured to provide reliable message exchange between Web services. Two of these policy sets (WS-I RSP and WS-I RSP ND) are immediately available, and the rest are readily available for import from a default repository.

Related tasks

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

"Defining and managing policy set bindings" on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

"Adding assured delivery to Web services through WS-ReliableMessaging" on page 634
WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services reliable without having to write custom code. You can get different qualities of service with WS-ReliableMessaging. These range from protecting against loss of messages across a network, through to protecting against a server becoming unavailable.

Using WS-Notification for publish and subscribe messaging for Web services

Related reference

"Service client policy set and bindings collection" on page 763
Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

Web services: Client security bindings collection
Use this page to view a list of application-level, client-side binding configurations for Web services security. These bindings are used when a Web service is a client to another Web service.

"Service providers collection at the cell level" on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Configuration tab: Configuration properties for this object. These property values are preserved even if the runtime environment is stopped then restarted. See the information center task descriptions for information about how to apply configuration changes to the runtime environment.

General Properties

Service client
Specifies the name of the service client that is displayed.

Additional Properties

Service integration bus
The system integration bus hosting this WS-Notification service Web service client.

WS-Notification service
The WS-Notification service hosting this client.

Policy Set Attachments

This section allows you to attach a policy set to the WS-Notification service client. You can complete the attachment by providing system-specific configuration when you assign the appropriate binding.

For more information about the policy set attachment options, see “Service client settings” on page 756.

Note: For WS-Notification service clients, policy set attachments are not supported at the endpoint (port) or operation level. Therefore, endpoints or operations are not selectable, and they are shown as inheriting any policy set or binding that is attached to the service client.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

About policy set bindings

In WebSphere Application Server Version 7.0, there are two types of bindings, application-specific bindings and general bindings.

Application specific binding

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can only create application-specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application-specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for WebSphere Application Server Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust

service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See “Defining and managing service client or provider bindings” on page 831. General provider policy set bindings might also be used for trust service attachments.

Buttons

Attach Client Policy Set	Click this button to view a list of policy sets available for attachment to the selected WS-Notification service client. Select a policy set from the list to attach the policy set to the selected service client. To close the drop-down list, click **Attach Client Policy Set**.
Detach Client Policy Set	Click this button to detach a policy set from a selected WS-Notification service client. After the policy set is detached, if there is no policy set attached at the upper level (that is, at the level of the WS-Notification service), the **Attached Client Policy Set** column displays `None` and the **Binding** column displays `Not Applicable`. If there is a policy set attached at the level of the WS-Notification service, the **Attached Client Policy Set** column displays *`policy_set_name`*`(inherited)` and the binding used for the upper level attachment is applied. The binding name is displayed with `(inherited)` after it.
Assign Binding	Click this button to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options: • Default • New Application Specific Binding For more information about these options, see “Service client settings” on page 756. To close the drop-down list, click Assign Binding.

Using WS-Transaction policy to coordinate transactions or business activities for Web services

You can use WS-Transaction policy to configure how a Java API for XML Web Services (JAX-WS) service or client handles Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) context.

About this task

The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the Web services environment. Distributed Web services applications, and the resources they use, can take part in distributed global transactions. With Web Services Business Activity (WS-BA) support in the application server, Web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll

back atomically, and therefore require a compensation process if an error occurs.

- Learn about WS-Transaction policy.
- Configure a JAX-WS client for WS-Transaction context.
- Configure a JAX-WS Web service for WS-Transaction context.
- Configure the WS-Transaction policy.
- Configure a WS-Transaction policy set using the wsadmin tool.
- Configure transaction properties for an application server.
- Configure the WS-Transaction specification level using the wsadmin tool.
- Configure WS-Transaction support in a secure environment.
- Configure an intermediary node for Web services transactions.
- Enable WebSphere Application Server to use an intermediary node for Web services transactions.
- Configure a server to use business activity support.
- Create an application that uses the Web Services Business Activity support.

Learning about WS-Transaction

WS-Transaction is an interoperability standard that includes the WS-AtomicTransaction, WS-BusinessActivity, and WS-Coordination specifications. Use these topics to learn more about WS-Transaction.

About this task

Note: In this release, when you work with policy sets, you can configure the WS-Transaction policy type for the WS-AtomicTransaction (WS-AT) and the WS-BusinessActivity (WS-BA) protocols. You can configure the way in which a Java API for XML Web Services (JAX-WS) client handles WS-AT or WS-BA context. You can specify that the client must send context, can send context if it is available, or must not send context.

The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the Web services environment. Distributed Web services applications, and the resources they use, can take part in distributed global transactions. With Web Services Business Activity (WS-BA) support in the application server, Web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs. Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant Web services can enlist to take part in the protocols that are offered by specific coordination types.

To learn about WS-Transaction in WebSphere Application Server, see the following topics:

- “Web Services Atomic Transaction support in the application server”
- “Web Services Business Activity support in the application server” on page 693
- “Web Services transactions, firewalls and intermediary nodes” on page 695
- “Transaction compensation and business activity support” on page 696
- “WS-Transaction and mixed-version cells” on page 701

Web Services Atomic Transaction support in the application server

The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the Web services environment. Distributed Web services applications, and the resources they use, can take part in distributed global transactions.

Web Services protocols provide standard ways of defining Web services applications, allowing the applications to operate independently of the product, platform, or programming language that is used. The

WS-AT support is an implementation of the following specifications on the application server. These specifications define a set of Web services that enable Web services applications to participate in global transactions that are distributed across the heterogeneous Web services environment.

- WS-AT is a specific coordination type that defines protocols for atomic transactions. The specifications are:
  - Web Services Atomic Transaction Version 1.0
  - Web Services Atomic Transaction Version 1.1
- Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant Web services can enlist to take part in the protocols that are offered by specific coordination types. The specifications are:
  - Web Services Coordination Version 1.0
  - Web Services Coordination Version 1.1

The WS-AT support is an interoperability protocol that introduces no new programming interfaces for transactional support. Global transaction demarcation is provided by standard enterprise application use of the Java Transaction API (JTA) UserTransaction interface. If an application component that is running under a global transaction makes a Web services request, a WS-AT CoordinationContext is implicitly propagated to the target Web service, but only if the appropriate application deployment descriptors have been set, as described in Configuring transactional deployment attributes.

If the application server is the system hosting the target endpoint for a Web services request that contains a WS-AT CoordinationContext, the application server automatically establishes a subordinate JTA transaction in the target runtime environment that becomes the transactional context under which the target Web services application runs.

The following figure, shows a transaction context shared between two application servers for a Web services request that contains a WS-AT CoordinationContext.

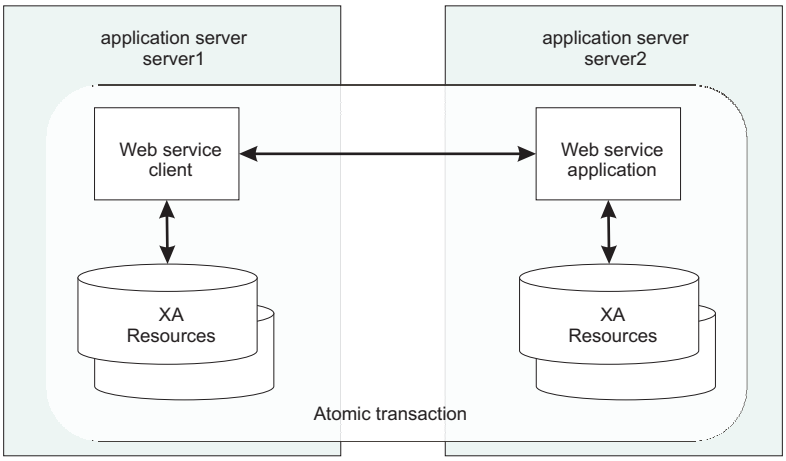


Figure 8. Transaction context shared between two application servers

You can configure the policies for the WS-AtomicTransaction protocol. You can configure whether a client propagates, and a server receives, a WS-AT context. To ensure that a client always sends WS-AtomicTransaction context when it makes an outbound service request, you must associate a policy set with the client, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Mandatory. Alternatively, if you know that the client always invokes remote endpoints that include the WS-AtomicTransaction ATAssertion policy type attribute, you can configure the client to apply the WS-Policy configuration of the provider so that the client adopts the mandatory policy of the provider automatically.

To ensure that any requests that a Web services provider receives include a WS-AtomicTransaction context, you must associate a policy set with the provider, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Mandatory.

To ensure that a client or provider never use WS-AtomicTransaction context, you must associate a policy set with the client or provider, where the policy set includes the WS-Transaction policy type, and this policy type must have a WS-AtomicTransaction setting of Never. You might use this configuration for environments where you do not want Web services requests to create a tight coupling between a client and a provider, for example when there are requests between enterprises.

If there is no policy set associated with a client or provider, or the WS-Transaction policy type is not included in the policy set, the default WS-Transaction behavior is used.

WS-AT support restrictions

In this version of the application server, WS-AT contexts cannot be started from a non-recoverable client process.

Application design considerations

WS-AT is a two-phase commit transaction protocol and is suitable for short duration transactions only.

An atomic transaction coordinates resource managers that isolate transactional updates by holding transactional locks on resources. Therefore, it is generally not recommended that WS-AT transactions are distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit, and in such scenarios, it can be more appropriate to use a compensating business transaction, for example, a Web Services Business Activity, or be part of a Business Process Execution Language (BPEL) process.

WS-AT is most appropriate for distributing transaction context across Web services that are deployed in a single enterprise. Only request-response message exchange patterns carry transaction context because the originator (application or container) of a transaction must be sure that all business tasks that are run under that transaction have finished before requesting the completion of a transaction. Web services invoked by a one-way request never run under the transaction of the requesting client.

The effect of service faults on WS-AT transactions is similar to the effect of Enterprise JavaBeans (EJB) application exceptions on transactions, as described in the EJB specification. If a service that is running under a requester's WS-AT transaction returns a fault, the application server does not automatically mark the transaction rollback-only. The exception handler of the requester decides whether the transaction can progress and chooses whether to mark the transaction rollback-only. If the requester is running in the application server, the standard JTA or EJB APIs can be used to mark the transaction rollback-only. The service component that generates the fault might, itself, mark the transaction rollback-only before returning the fault. If the implementation of the service component encounters a system exception, it typically allows its container to handle the exception. Application server containers automatically mark any received transaction context rollback-only when handling a system exception that is generated by a service implementation.

Application development considerations

There are no specific development tasks required for Web services applications to take advantage of WS-AT.

For Java API for XML-based RPC (JAX-RPC) applications, there are some application deployment descriptors that you must set appropriately, as described in Configuring transactional deployment attributes. The JAX-RPC run time supports WS-AT 1.0.

Note: WebSphere Application Server supports both the WS-Transaction 1.1 and the WS-Transaction 1.0 specifications. You can configure the default WS-Transaction specification level to use for outbound requests if the specification level that the server requires cannot be determined from the provider policy. This applies to outbound requests that include a Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) coordination context.

For Java API for XML-Based Web Services (JAX-WS) applications, enable WS-AT support by creating a policy set, adding the WS-Transaction policy type to the policy set, optionally configuring the policy type, and attaching the policy set to the application or client that will be involved in the WS-AT communication. The JAX-WS run time supports WS-AT 1.0, WS-AT 1.1 and the WS-Policy assertion for WS-AT.

When the JAX-WS runtime receives an inbound request, both WS-Transaction 1.0 and WS-Transaction 1.1 specification levels are supported. When an outbound JAX-WS request is sent, only one specification level can be used. If WS-Transaction WS-Policy assertions are available, either from the Web Services Description Language (WSDL) of the target Web service, or from the WS-Transaction policy type of the client, the specification level that is applicable to the client and the target Web service is used. For example, if the hosting environment of the target Web service supports only WS-Transaction 1.0, WS-AT 1.0 is used. If both specification levels are applicable, or if no WS-Transaction WS-Policy assertions are available, the default WS-Transaction specification level that is set in the Transaction service settings is used.

To move a JAX-WS application that was using WS-AT 1.0 to use WS-AT 1.1, alter the policy set that is attached to the application; you do not need to change the application.

The default behavior when there is no effective policy set, or when the WS-Transaction policy type is not included in the effective policy set, is as follows:

- For a client, if the client does not consider the policy of the provider, the client does not send any WS-AT or Web Services Business Activity (WS-BA) context. This behavior is equivalent to a WS-Transaction policy setting of Never.
- For a client, if the client considers the policy of the provider, the client sends WS-AT or WS-BA context if the policy of the provider includes WS-AT or WS-BA assertions. This behavior is equivalent to a WS-Transaction policy setting of Supports.
- For a server, the server does not receive any WS-AT or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.

Application developers do not need to explicitly register WS-AT participants. The application server run time takes responsibility for the registration of WS-AT participants, in the same way as the registration of XAResources in the JTA transaction to which the WS-AT transaction is federated. At transaction completion time, all XAResources and WS-AT participants are coordinated atomically by the application server transaction service.

If a JTA transaction is active on the thread when a Web services application request is made, the transaction is propagated across on the Web services request and established in the target environment. This process is similar to the distribution of transaction context over IIOP, as described in the EJB specification. Any transactional work performed in the target environment becomes part of the same global transaction.

WS-Transaction policy assertions

If you configure the policies for the WS-Transaction protocol for a provider, this configuration affects the assertions that are included in any WSDL that is generated for the Web service with which the policy type is associated. The WS-Policy assertion that is used to describe the transactional requirements of a client or provider that uses WS-AtomicTransaction is ATAssertion. If the WS-Transaction policy type has a WS-AtomicTransaction setting of Mandatory or Supports, a policy assertion is included in the WSDL.

The application server can also parse, understand, and apply such assertions that are in WSDL that it parses.

The following example shows WSDL where the WS-AtomicTransaction ATAssertion indicates that an endpoint must be invoked with WS-AT context included in the request message, and that the context can be in WS-Transaction 1.0 or 1.1 format. There are two namespaces and there are two assertions; one for each WS-Transaction specification level, using the WS-Policy ExactlyOne operator to show that the client must choose which specification level to use.

```
<wsdl:definitions targetNamespace="bank.example.com"
     xmlns:tns="bank.example.com"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wsat11="http://docs.oasis-open.org/ws-tx/wsat/2006/06"
     xmlns:wsat10="http://schemas.xmlsoap.org/ws/2004/10/wsat"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <wsp:Policy wsu:Id="ATPolicy">
         <wsp:ExactlyOne>
             <wsat11:ATAssertion />
             <wsat10:ATAssertion />
             <!-- omitted assertions -->
         </wsp:ExactlyOne />
     </wsp:Policy>
     <!-- omitted elements -->
     <wsdl:binding name="BankBinding" type="tns:BankPortType">
          <!-- omitted elements -->
          <wsdl:operation name="TransferFunds">
               <wsp:PolicyReference URI="#ATPolicy" wsdl:required="true"/>
               <!-- omitted elements -->
          </wsdl:operation>
     </wsdl:binding>
</wsdl:definitions>
```

Web Services Business Activity support in the application server

With Web Services Business Activity (WS-BA) support in the application server, Web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs.

Web Services protocols are defined by the Organization for the Advancement of Structured Information Standards (OASIS) group and provide standard ways of defining Web services applications, allowing the applications to operate independently of the product, platform or programming language that is used. The Web Services Business Activity (WS-BA) support is an implementation of the following specifications in the application server. These specifications define a set of protocols that enable Web services applications to participate in loosely coupled business processes that are distributed across the heterogeneous Web services environment, with the ability to compensate actions if an error occurs. For example, an application that sends an e-mail cannot retrieve that e-mail following a failure in the business task. However, the application can provide a business-level compensation handler that sends another e-mail advising of the new circumstances. A *business activity* is a group of general tasks that you want to link together so that the tasks have an agreed outcome.

- WS-BA is a specific coordination type that defines protocols for business activities. The specifications are:
  - Web Services Business Activity Version 1.0
  - Web Services Business Activity Version 1.1
- Web Services Coordination (WS-COOR) specifies a CoordinationContext and a Registration service with which participant Web services can enlist to take part in the protocols that are offered by specific coordination types. The specifications are:
  - Web Services Coordination Version 1.0
  - Web Services Coordination Version 1.1

In addition to supporting the WS-BA interoperability protocol, the application server provides a programming interface for creating business activities and compensation handlers. With this programming interface, you can specify compensation data and check or alter the status of a business activity.

You can also use this compensation facility with applications that are not Web services, as long as these applications involve communication between WebSphere Application Servers only. See the related topics for more information.

You can configure the policies for the WS-BusinessActivity protocol. You can configure whether a client propagates, and a server receives, a WS-BA context. To ensure that a client always sends WS-BusinessActivity context when it makes an outbound service request, you must associate a policy set with the client, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Mandatory. Alternatively, if you know that the client always invokes remote endpoints that include the WS-BusinessActivity BAAtomicOutcomeAssertion policy type attribute, you can configure the client to apply the WS-Policy configuration of the provider so that the client adopts the mandatory policy of the provider automatically.

To ensure that any requests that a Web services provider receives includes a WS-BusinessActivity context, you must associate a policy set with the provider, where the policy set must include the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Mandatory.

To ensure that a client or provider never use WS-BusinessActivity context, you must associate a policy set with the client or provider, where the policy set includes the WS-Transaction policy type, and this policy type must have a WS-BusinessActivity setting of Never. You might use this configuration for environments where you do not want Web services requests to create a tight coupling between a client and a provider, for example when there are requests between enterprises.

If no policy set is associated with a client or provider, or the WS-Transaction policy type is not included in the policy set, the default WS-Transaction behavior is used.

Application development considerations

No specific development tasks are required for Web services applications to take advantage of WS-BA.

For Java API for XML-based RPC (JAX-RPC) applications, any Enterprise JavaBeans (EJB) component that is configured to run under a BusinessActivity scope automatically propagates that scope when it makes an outbound JAX-RPC Web services request. The JAX-RPC run time supports WS-BA 1.0.

For Java API for XML-Based Web Services (JAX-WS) applications, enable WS-BA support by creating a policy set, adding the WS-Transaction policy type to the policy set, optionally configuring the policy type, and attaching the policy set to the application or client that will be involved in the WS-BA communication. The JAX-WS run time supports WS-BA 1.0, WS-BA 1.1, and the WS-Policy assertion for WS-BA.

When the JAX-WS run time receives an inbound request, both WS-Transaction 1.0 and WS-Transaction 1.1 specification levels are supported. When an outbound JAX-WS request is sent, only one specification level can be used. If WS-Transaction WS-Policy assertions are available, either from the Web Services Description Language (WSDL) of the target Web service, or from the WS-Transaction policy type of the client, the specification level that is applicable to the client and the target Web service is used. For example, if the hosting environment of the target Web service supports only WS-Transaction 1.0, WS-BA 1.0 is used. If both specification levels are applicable, or if no WS-Transaction WS-Policy assertions are available, the default WS-Transaction specification level that is set in the Transaction service settings is used.

The default behavior when there is no effective policy set, or when the WS-Transaction policy type is not included in the effective policy set, is as follows:

- If a client does not consider the policy of the provider, the client does not send any Web Service Atomic Transaction (WS-AT) or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.
- If a client does consider the policy of the provider, the client sends WS-AT or WS-BA context if the policy of the provider includes WS-AT or WS-BA assertions. This behavior is equivalent to a WS-Transaction policy configuration setting of Supports.
- A server does not receive any WS-AT or WS-BA context. This behavior is equivalent to a WS-Transaction policy configuration setting of Never.

WS-Transaction policy assertions

If you configure the policies for the WS-BusinessActivity protocol for a provider, this affects the assertions that are includes in any WSDL that is generated for the Web service with which the policy type is associated. The WS-Policy assertion that is used to describe the compensation requirements of a client or provider that uses WS-BusinessActivity is BAAtomicOutcomeAssertion. If the WS-Transaction policy type has a WS-BusinessActivity setting of Mandatory or Supports, a policy assertion is included in the WSDL.

The application server can also parse, understand, and apply such assertions that are in WSDL that it parses.

The following example shows WSDL where the WS-BusinessActivity BAAtomicOutcomeAssertion indicates that an endpoint must be invoked with WS-BA context included in the request message, and that the context can be in WS-Transaction 1.0 or 1.1 format. There are two namespaces and two assertions; one for each WS-Transaction specification level, using the WS-Policy ExactlyOne operator to show that the client must choose which specification level to use.

```
<wsdl:definitions targetNamespace="bank.example.com"
     xmlns:tns="bank.example.com"
     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
     xmlns:wsat11="http://docs.oasis-open.org/ws-tx/wsba/2006/06"
     xmlns:wsat10="http://schemas.xmlsoap.org/ws/2004/10/wsba"
     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
     oasis-200401-wss-wssecurity-utility-1.0.xsd">
     <wsp:Policy wsu:Id="BAPolicy">
         <wsp:ExactlyOne>
             <wsat11:BAAtomicOutcomeAssertion />
             <wsat10:BAAtomicOutcomeAssertion />
             <!-- omitted assertions -->
         </wsp:ExactlyOne />
     </wsp:Policy>
     <!-- omitted elements -->
     <wsdl:binding name="BankBinding" type="tns:BankPortType">
          <!-- omitted elements -->
          <wsdl:operation name="TransferFunds">
               <wsp:PolicyReference URI="#BAPolicy" wsdl:required="true"/>
               <!-- omitted elements -->
          </wsdl:operation>
     </wsdl:binding>
</wsdl:definitions>
```

Web Services transactions, firewalls and intermediary nodes

You can configure your system to enable propagation of Web Services Atomic Transactions (WS-AT) message contexts and Web Service Business Activities (WS-BA) message contexts across firewalls or outside the WebSphere Application Server domain. With these configurations, you can distribute Web service applications that use WS-AT or WS-BA across disparate systems.

The topologies that are available to you are as follows:

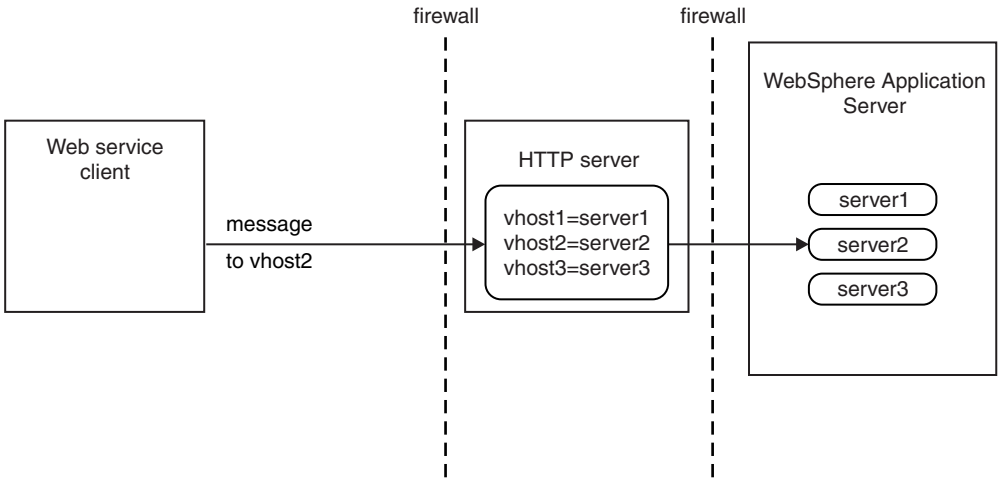
Direct connection

No intermediary node exists in this topology. The client communicates directly with the specific WebSphere Application Server on which the target service resides.

HTTP server, such as IBM HTTP Server

In this topology, the client communicates with an HTTP server, which always routes the client requests and Web services transaction protocol messages to a specific WebSphere Application Server. Also, you must configure the HTTP server for Web services transactions, that is, configure it to deliver Web services transaction protocol messages to the appropriate WebSphere Application Server.

Transactional integrity is assured, because recovery processing occurs after the failed server restarts.



Transaction compensation and business activity support

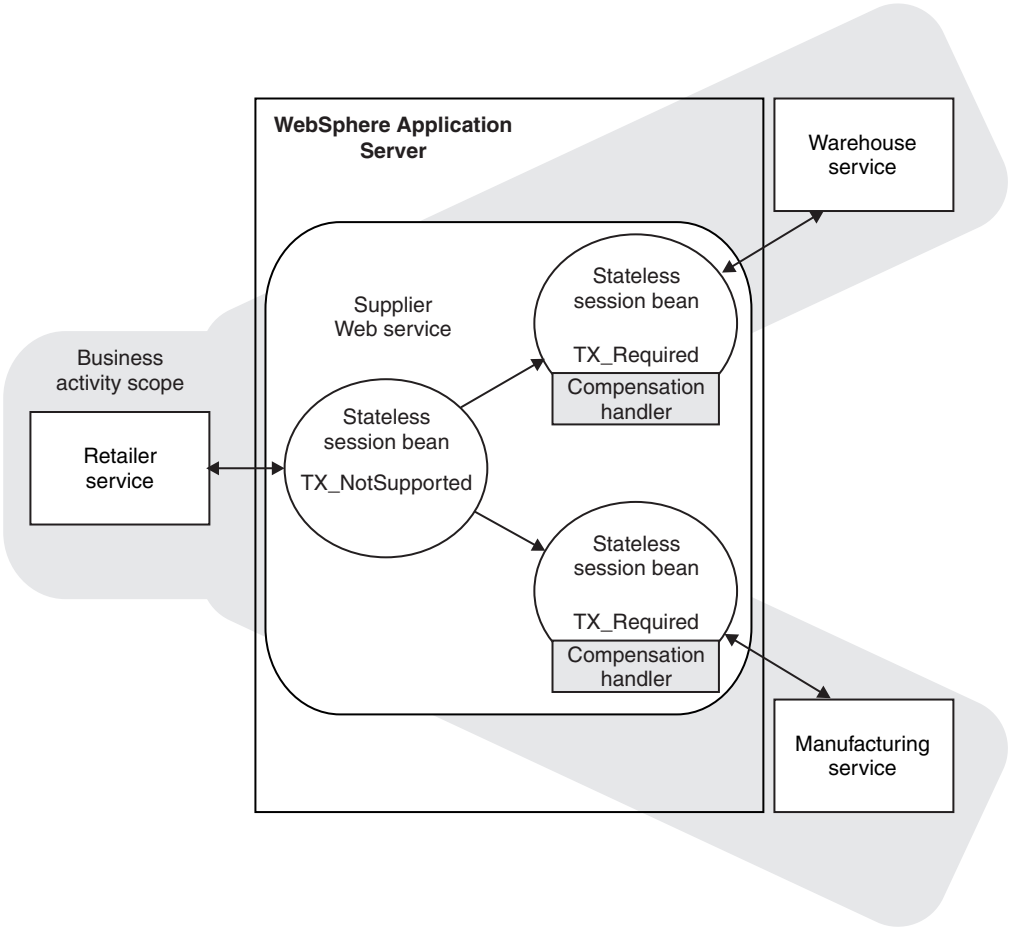
A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an e-mail can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple Web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Server environments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated with the Warehouse and Manufacturing services, and that perform work that can be compensated. These other session beans each have a *compensation handler*, a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an e-mail.



Application design considerations

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functionality. Enterprise beans that exploit business activity scopes can offer Web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in Web service messages using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as Web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment considerations

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

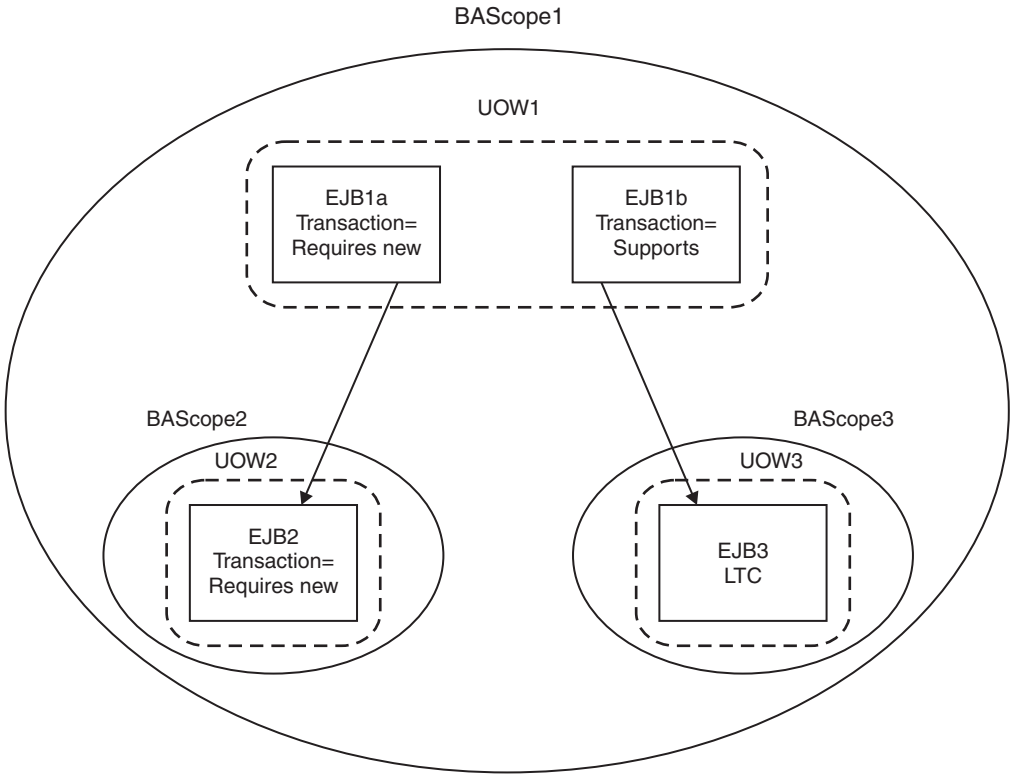
Business activity scopes

The scope of a business activity is that of a core WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing core UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any core UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.



Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by retrying the called component or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 5. Compensation behavior for a single business activity scope

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see “Business activity API” on page 702.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration
: If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope
: Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method,

then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For instructions on how to perform these actions, see “Creating an application that uses the Web Services Business Activity support” on page 711.

WS-Transaction and mixed-version cells

Considerations exist for WS-Transaction policy type enablement and behavior, and the WS-Transaction specification level to use, when a cell contains servers at different versions; for example, WebSphere Application Server Version 7.0 and WebSphere Application Server Version 6.1 Feature Pack for Web Services.

WS-Transaction policy type enablement

For a Version 6.1 Feature Pack for Web Services server, you can enable the WS-Transaction policy type by including it in a policy set, but you cannot configure it. For a Version 7.0 server, you can both enable and configure the WS-Transaction policy type. Configuration information is written to the WS-Transaction policy type file.

In a cell with both Version 6.1 Feature Pack for Web Services and Version 7.0 servers, the following behavior occurs:

- If a Version 6.1 Feature Pack for Web Services server reads a WS-Transaction policy type file that is generated by a Version 7.0 server, the server enables the WS-Transaction policy type, but ignores any configuration information in the file.
- If a Version 7.0 server reads a WS-Transaction policy type that is generated by a Version 6.1 Feature Pack for Web Services server, the server enables the WS-Transaction policy type using a value of Supports for the WS-AtomicTransaction and WS-BusinessActivity protocols. This value is equivalent to the existing behavior of a Version 6.1 Feature Pack for Web Services server.

WS-Transaction specification level

A WebSphere Application Server Version 6.x server supports WS-Transaction 1.0. A Version 7.0 server supports WS-Transaction 1.0 and 1.1.

No special considerations exist for a cell with both Version 6.x and Version 7.0 servers.

Related concepts

“Using WS-Transaction policy to coordinate transactions or business activities for Web services” on page 688
You can use WS-Transaction policy to configure how a Java API for XML Web Services (JAX-WS) service or client handles Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) context.

Highly available messaging engine configuration

Related reference

“WS-Transaction policy settings” on page 932
Use this page to specify the policies for the WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA) protocols. WS-AT supports coordination of activities so that either all the activities occur, or none of them occur. WS-BA supports coordination of activities that are more loosely coupled than atomic transactions, and that therefore, require a compensation process if an error occurs.

“Transaction service settings” on page 2145
Use this page to specify settings for the transaction service. The transaction service is a server runtime component that can coordinate updates to multiple resource managers to ensure atomic updates of data. Transactions are started and ended by applications or the container in which the applications are deployed.

Business activity API

Use the business activity application programming interface (API) to create business activities and compensation handlers for an application component, and to log data that is required to compensate an activity if there is a failure in the overall business activity.

Overview

The business activity support provides a UserBusinessActivity API and two interfaces: a serializable.CompensationHandler interface and a CompensationHandler interface. Each interface has two exceptions: RetryCompensationHandlerException and CompensationHandlerFailedException. You can look up the UserBusinessActivity interface from the application server Java Naming and Directory Interface (JNDI) at `java:comp/websphere/UserBusinessActivity`. For example:

```
InitialContext ctx = new InitialContext();
UserBusinessActivity uba = (UserBusinessActivity) ctx.lookup("java:comp/websphere/UserBusinessActivity");
```

You can use the getId method to access the unique identifier for the business activity that is currently associated with the calling thread. The identifier is the same as the one that is generated for the business activity scope at run time and that is used for information, warning, and error messages. For example, the application can use the identifier in audit or diagnostic messages, and it is possible to correlate between application-generated and runtime-generated messages.

```
InitialContext initialContext = new InitialContext();
UserBusinessActivity uba = initialContext.lookup("java:comp/websphere/UserBusinessActivity");
...
String activityId = uba.getId();
if (activityId == null)
// No activity on the thread
else
// Output audit message including activity id
```

If an application component runs work that might require compensating upon failure in the business activity, you must provide a compensation handler class that is assembled as part of the deployed application. This Java class must implement one of the following interfaces:

- com.ibm.websphere.wsba.serializable.CompensationHandler, which takes a takes a parameter of a serializable object
- com.ibm.websphere.wsba.CompensationHandler, which takes a parameter of a Service Data Object (SDO)

Typically, applications that already have their data available in DataObject format will use the CompensationHandler interface, and applications that do not will use the serializable.CompensationHandler interface. Both interfaces support the close and compensate methods.

An application must register a compensation handler implementation that works with the type of compensation data (serializable object or SDO) that the application uses. If there is a mismatch between the type of data that the application component uses and the compensation handler implementation, there is an error.

During normal application processing, the application can make one or more invocations to the `setCompensationDataImmediate` or `setCompensationDataAtCommit` methods, passing in either a serializable object or an SDO that represents the current state of the work performed.

When the underlying unit of work (UOW) that the root business activity is associated with completes, all registered compensators are coordinated to complete. During completion, either the compensate or the close method is called on the compensation handler, passing in the most recent compensation data logged by the application component as a parameter. Your compensation handler implementation must be able to understand the data that is stored in either the serializable object or the SDO DataObject; using this data, the compensation handler must be able to determine the nature of the work performed by the enterprise bean and compensate or close in an appropriate way, for example by undoing changes made to database rows if there is a failure in the business activity. You associate the compensation handler with an application component by using the assembly tooling, such as Rational Application Developer.

Active and inactive compensation handlers

You implement the serializable.CompensationHandler or CompensationHandler interface for any application component that executes code that might need to be compensated within a business activity scope. Compensation handler objects are registered implicitly with the business activity scope under which the application runs, whenever the application calls the UserBusinessActivity API to specify compensation data. Compensation handlers can be in one of two states, active or inactive, depending on any transactional UOW under which they are registered. A compensation handler that is registered within a transactional UOW might initially be inactive until the transaction commits, at which point the compensation handler becomes active (see the following section). A compensation handler that is registered outside a transactional UOW always becomes active immediately.

When a business activity completes, it drives only active compensation handlers. Any inactive compensation handlers that are associated with the business activity are discarded and never driven.

Logging compensation data

The business activity API specifies two methods that allow the application to log compensation data. This data is made available to the compensation handlers during their processing when the business activity completes. The application calls one of these methods, depending on whether it expects transactions to be part of the business activity.

`setCompensationDataAtCommit()`

Call the setCompensationDataAtCommit method when the application expects a global transaction on the thread.

- If a global transaction is present on the thread, the CompensationHandler object is initially inactive. If the global transaction fails, it rolls back any transactional work done within its transaction context in an atomic manner, and drives the business activity to compensate other completed UOWs. The compensation handler does not need to be involved. If the global transaction commits successfully, the compensation handler becomes active because if the overall business activity fails, the compensation handler is required to compensate the durable work that is completed by the global transaction. The setCompensationDataAtCommit method configures the CompensationHandler instance to perform this compensation function.

- If a global transaction is not present when the setCompensationDataAtCommit method is called, the compensation handler becomes active immediately.

For example, for an SDO, using the same business activity instance as in the previous example:

```
DataObject compensationData = doWorkWhichWouldNeedCompensating();
uba.setCompensationDataAtCommit(compensationData);
```

setCompensationDataImmediate()

Call the setCompensationDataImmediate method when the application does not expect a global transaction on the thread.

The setCompensationDataImmediate method makes a CompensationHandler instance active immediately, regardless of the current UOW context at the time that the method is invoked. The compensation handler is always able to participate during completion of the business activity.

The role of the setCompensationDataImmediate method is to compensate any non-transactional work, in other words, work that can be performed either inside or outside a global transaction, but that is not governed by the transaction. An example of this type of work is sending an e-mail. The compensation handler must be active immediately so that if a failure occurs in a business activity, this non-transactional work is always compensated.

For example, for a serializable object, using the same business activity instance as in the previous example:

```
Serializable compensationData = new MyCompensationData();
uba.setCompensationDataImmediate(compensationData);
```

Although these two compensation data logging methods, if called in the same enterprise bean, use the same compensation handler class, they create two separate instances of the compensation handler class at run time. Therefore, the actions of the methods are mutually exclusive; calling one of the methods does not overwrite any work carried out by the other method.

If a compensation handler instance is already added to the Business Activity using one of these methods, and then the same method is called, passing in `null` as a parameter, that compensation handler instance is removed from the business activity, and is not driven to close or compensate during completion of the business activity.

As described previously, the business activity support adds a compensation handler instance to the business activity when a compensation data logging method is called for the first time by the enterprise bean that uses that business activity. At the same time, a snapshot of the enterprise application context is taken and logged with the compensation data. When the business activity competes, all the compensation handlers that were added to the business activity are driven to compensate or close. The code that you create in the CompensationHandler or serializable.CompensationHandler class is guaranteed to run in the same enterprise application context that was captured in the earlier snapshot.

For details about the methods available in the business activity API, see Additional Application Programming Interfaces (APIs).

Configuring a JAX-WS client for WS-Transaction context

You can configure the way that a Java API for XML Web Services (JAX-WS) client handles Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) context by configuring the Web Services Transaction (WS-Transaction) policy type. You can specify that the client must send context, can send context if it is available, or must not send context.

Before you begin

A JAX-WS client must be installed.

About this task

You can configure a WS-Transaction policy set using the administrative console, as described in this task, or you can configure a WS-Transaction policy set using the wsadmin tool.

To configure a JAX-WS client for WS-Transaction context using the administrative console, complete the following steps.

1. Create a new policy set, or copy and rename an existing policy set. You can copy an existing user-defined policy set, or one of the WS-Transaction default policy sets (WSTransaction or SSL WSTransaction). See “Creating policy sets using the administrative console” on page 806.
2. Check that your policy set includes the WS-Transaction policy type. If necessary, add the WS-Transaction policy type. See “Adding policies to policy sets using the administrative console” on page 903.
3. Configure the WS-Transaction policy.
4. Associate the policy set with the JAX-WS client. See Managing policy sets and bindings for service clients.
5. Choose a WS-Policy application rule that includes the configured policy of the client, that is, client only or client and provider. See “Configuring the client policy using a service provider policy” on page 724.
6. Save your changes to the master configuration.

What to do next

The JAX-WS client is configured to use WS-Transaction context in the way that you specified.

Configuring a JAX-WS Web service for WS-Transaction context

You can configure the way that a Java API for XML Web Services (JAX-WS) service handles Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) context by configuring the Web Services Transaction (WS-Transaction) policy type. You can specify that the Web service must receive context, can receive context if it is available, or must not receive context.

Before you begin

A JAX-WS client must be installed.

About this task

You can configure a WS-Transaction policy set using the administrative console as described in this task, or you can configure a WS-Transaction policy set using the wsadmin tool.

To configure a JAX-WS service for WS-Transaction context using the administrative console, complete the following steps.

1. Create a new policy set, or copy and rename an existing policy set. You can copy an existing user-defined policy set, or one of the WS-Transaction default policy sets (WSTransaction or SSL WSTransaction). See “Creating policy sets using the administrative console” on page 806.
2. Check that your policy set includes the WS-Transaction policy type. If necessary, add the WS-Transaction policy type. See “Adding policies to policy sets using the administrative console” on page 903.
3. Configure the WS-Transaction policy.
4. Associate the policy set with the JAX-WS Web service, endpoint, or operation for which the specified behavior is required. See Managing policy sets and bindings for service providers.
5. Save your changes to the master configuration.

What to do next

The JAX-WS Web service, endpoint, or operation is configured to use WS-Transaction context in the way that you specified.

Configuring a WS-Transaction policy set using the wsadmin tool

You can configure the way that a Java API for XML Web Services (JAX-WS) client or Web service handles Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) context by configuring the Web Services Transaction (WS-Transaction) policy type. You can specify that the client or service must use context, can use context if it is available, or must not use context. Use command scripts to configure a policy set for Web services transactions.

About this task

You can configure a WS-Transaction policy set using the wsadmin tool as described in this task, or you can configure a WS-Transaction policy set using the administrative console.

1. Start the wsadmin tool if it is not already running.
2. Use the createPolicySet command to create a new policy set, or the copyPolicySet command to copy and rename an existing policy set. You can copy an existing user-defined policy set, or one of the WS-Transaction default policy sets (WSTransaction or SSL WSTransaction).
3. Check that your policy set includes the WS-Transaction policy type. If necessary, add the WS-Transaction policy type. For example:

```
AdminTask.importPolicySet('[-defaultPolicySet WSTransaction]')
AdminTask.addPolicyType('[-policySet PolicySet1
-policyType WSTransaction -enabled true]')
```

4. Use the setPolicyType command to configure the WS-Transaction policy type attributes. For example:

```
AdminTask.setPolicyType('[-policySet PolicySet1
-policyType WSTransaction
-attributes "[ [BAAtomicOutcomeAssertion mandatory] [ATAssertion supports] ]"]
-replace')
```

   For detailed information about these configurable attributes, see “WS-Transaction policy settings” on page 932.

5. Save your changes to the master configuration. For example, enter the following command:

```
AdminConfig.save()
```

What to do next

You are now ready to associate the policy set with the JAX-WS client, or with the JAX-WS Web service, endpoint, or operation.

Configuring Web Services Transaction support in a secure environment

If you use Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) support when administrative security is enabled, you might need to change the default transaction service configuration. You can disable the transaction coordination authorization setting, create a new Web container transport chain, or do both.

About this task

You might disable transaction coordination authorization if you want to interoperate with other servers and you do not want to set up security for the transaction manager to support the Common Criteria EAL4 evaluated configuration. When transaction coordination authorization is disabled, WebSphere Application Server does not automatically reject secure WS-Transactions protocol messages.

You might configure a new Web container transport chain for use by WS-Transactions in the following situations:

- You want to use an alternative port number for WS-AT or WS-BA protocol messages.
- You want to interoperate with a non-WebSphere Application Server product that requires client certificate authentication on the Secure Sockets Layer (SSL) connection that is used for protocol messages.

The transaction service, by default, selects a suitable Web container transport chain from the list of those configured and uses it for protocol messages. You can configure a new transport chain and specify your own settings. For example, you can specify an alternative SSL configuration that requires client certificate authentication, which is then used specifically for WS-Transactions protocol messages.

1. Optionally, use the following steps to disable transaction coordination authorization.
   a. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Container Settings] Container Services** → **Transaction Service**.
   b. Clear the **Enable transaction coordination authorization** check box.
   c. Click **Apply** or **OK**.
   d. Save your changes to the master configuration.
2. Optionally, use the following steps to create a new Web container transport chain.
   a. In the administrative console, click **Servers** → **Application servers** → ***server_name*** → **[Container Settings] Web Container Settings** → **Web container transport chains**.
   b. Click **New** to create a new transport chain.
   c. Type a name for the transport chain.
   d. From the Transport chain template list, select an appropriate template.
   e. Click **Next** to select a new port for the chain.
   f. Type a name, host, and port number for the port. For a secure chain, the host must match the common name in the certificate that is used.
   g. Click **Next**, confirm the settings, then click **Finish**.
   h. Save your changes to the master configuration.
   i. If necessary, create a new SSL configuration and associate it with the SSL channel associated with your new chain. For more information, see Creating a Secure Sockets Layer configuration. You are now ready to configure the transaction service to use the new transport chain.
   j. Click **Servers** → **Application servers** → ***server_name*** → **[Container Settings] Container Services** → **Transaction Service**.
   k. In the External WS-Transaction HTTP(S) URL prefix section, click **Select prefix**, then select the Web container transport chain that you have just created from the list.

      If you are using an intermediary, such an HTTP proxy, in front of the application server, click **Specify custom prefix**, then type the external endpoint URL information for the intermediary node in the field. For more information, see “Enabling WebSphere Application Server to use an intermediary node for Web services transactions” on page 709.
   l. Click **Apply** or **OK**, then save your changes to the master configuration.
3. After you save all the configuration changes, restart the server for the changes to take effect.

Results

You configured your system to use WS-AT or WS-BA in a secure environment.

Configuring an intermediary node for Web services transactions

Intermediary nodes allow the exchange of Web Services Atomic Transaction (WS-AT) and Web Services Business Activity (WS-BA) protocol messages across firewalls and outside the WebSphere Application Server domain. You configure an intermediary node to specify which WebSphere Application servers the node routes requests to.

1. Configure the HTTP server so that it routes WS-AT and WS-BA requests that are targeted at WebSphere Application Server to WebSphere Application Server, rather than processing them itself.
2.
3.

Results

You configured the intermediary node ready for use by WebSphere Application Server.

Example

What to do next

Configure WebSphere Application Server to use the intermediary node by specifying, for each server, the appropriate virtual host of the intermediary node.

Example: Configuring IBM HTTP server as an intermediary node for Web services transactions

You can use an HTTP server intermediary nodes to enable the exchange of Web Services Atomic Transaction and Web Services Business Activity protocol messages across firewalls and outside the WebSphere Application Server domain. For IBM HTTP server, you achieve this behavior by modifying the plugin-cfg.xml file of the IBM HTTP server node.

Routing requests to WebSphere Application Server

You can use the IBM HTTP server as a single intermediary node, or you can combine it with a Proxy Server for IBM WebSphere Application Server. In both cases, update the plugin-cfg.xml file to indicate that the HTTP server should route requests that are targeted at WebSphere Application Server, those of the form http://*host:port*/_IBMSYSAPP/*, to WebSphere Application Server, rather than processing them itself.

To update the plugin-cfg.xml file, add a URI element with a name of _IBMSYSAPP, as shown in the following example. Add this URI to all UriGroup elements in the plugin-cfg.xml file.

```
<UriGroup Name="default_host_server1_99T73NKNode01_Cluster_URIs">
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hello" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hitcount" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsp" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsv" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="*.jsw" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/j_security_check" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/ibm_security_logout" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/servlet/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/SamplesGallery/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/WSsamples/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/PlantsByWebSphere/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/PlantsByWebSphere/docs/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/_IBMSYSAPP/*" />
</UriGroup>
```

Configuring virtual host mapping

If you are using IBM HTTP as the only intermediary node, in other words you are not also using Proxy Server for IBM WebSphere Application Server, configure virtual hosts to represent each WebSphere Application Server that the HTTP node routes requests to. Update the plugin-cfg.xml file by adding VirtualHostGroup, VirtualHost and Route elements.

The following example shows part of the plugin-cfg.xml file for a configuration in which the IBM HTTP server routes requests to one of two servers, server1 and server2, in WebSphere Application Server.

The plugin-cfg.xml file contains two virtual host aliases, with names name1.acme.com and name2.acme.com, that are defined using VirtualHost and VirtualHostGroup elements. The Route elements define the association between the virtual hosts and the ServerCluster elements. When a request is made, IBM HTTP server finds the best matching route to dispatch the request to. A request made to virtual host name1.acme.com, with a URI that matches a pattern in the default_URIs URI group, is sent to the server1_Cluster server cluster. This server cluster contains only one server, server1, so requests targeted at virtual host name1.acme.com are sent to server1, and similarly, requests targeted at virtual host name2.acme.com are sent to server2.

```
<UriGroup Name="default_URIs">
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/snoop/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hello" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/hitcount" />
...
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/PlantsByWebSphere/*" />
<Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/PlantsByWebSphere/docs/*" />
  <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/_IBMSYSAPP/*" />
</UriGroup>

<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin" Name="server1_Cluster" PostBufferSize="64"
PostSizeLimit="-1" RemoveSpecialHeaders="true" RetryInterval="60">
      <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="server1" ServerIOTimeout="0" WaitForContinue="false">
         ...
      </Server>
      <PrimaryServers> <Server Name="server1"/> </PrimaryServers>
   </ServerCluster>
<ServerCluster CloneSeparatorChange="false" LoadBalance="Round Robin" Name="server2_Cluster" PostBufferSize="64" PostSizeLimit="-1" RemoveSpecialHeaders="true" R
      <Server ConnectTimeout="0" ExtendedHandshake="false" MaxConnections="-1" Name="server2" ServerIOTimeout="0" WaitForContinue="false">
         ...
     </Server>
      <PrimaryServers> <Server Name="server2"/>  </PrimaryServers>
</ServerCluster>
<VirtualHostGroup Name="vhost_server1"> <VirtualHost Name="name1.acme.com:9081"/> </VirtualHostGroup>
<VirtualHostGroup Name="vhost_server2"> <VirtualHost Name="name2.acme.com:9081"/> </VirtualHostGroup>
<Route ServerCluster="server1_Cluster" UriGroup="default_URIs" VirtualHostGroup=" vhost_server1 "/>
<Route ServerCluster="server2_Cluster" UriGroup="default_URIs" VirtualHostGroup=" vhost_server2"/>
```

Related concepts

“Web Services transactions, firewalls and intermediary nodes” on page 695
You can configure your system to enable propagation of Web Services Atomic Transactions (WS-AT) message contexts and Web Service Business Activities (WS-BA) message contexts across firewalls or outside the WebSphere Application Server domain. With these configurations, you can distribute Web service applications that use WS-AT or WS-BA across disparate systems.

Enabling WebSphere Application Server to use an intermediary node for Web services transactions

You can use intermediary nodes with Web Services Atomic Transactions (WS-AT) or Web Services Business Activities (WS-BA) to support the exchange of associated requests across firewalls and outside the WebSphere Application Server domain. You configure WebSphere Application Server to use an intermediary node by specifying the external endpoint URL information for the intermediary node in each server that is accessed through the intermediary.

Before you begin

Configure the intermediary node that you want to use, and ensure that you know the address or addresses of the intermediary node that you want to map to servers in your WebSphere Application Server configuration.

Configure the intermediary to listen on a specific port for protocol messages and to route these messages to the specific WebSphere Application Server instance that you want to enable. See the related information for an example configuration where an IBM HTTP server is the intermediary.

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Container Services] Transaction Service**.

2. In the External WS-Transaction HTTP(S) URL prefix section, click **Specify custom prefix**, then type the external endpoint URL information for the intermediary node in the field. Use one of the following formats for the prefix, where *host_name* and *port* represent the intermediary node that is an HTTP or HTTPS proxy for the server, and *port* is optional.
   - `http://host_name:port`
   - `https://host_name:port`
3. Click **Apply** or **OK**.
4. Save your changes to the master configuration.
5. Repeat the previous steps for each server that is accessed through the intermediary node.
6. Restart the servers.

Results

You configured your system to use an intermediary node. Test your configuration to ensure that messages are routed as you expect.

Configuring a server to use business activity support

Business activity support provides compensation for activities such as sending an e-mail, which can be difficult or impossible to roll back atomically. With this compensation, applications on disparate systems can coordinate activities that are more loosely coupled than atomic transactions. To use the business activity support, you must first enable it on each server that you plan to use.

About this task

If an application component uses business activity support, you must enable the support on each server that runs the application.

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Container Settings] Container Services** → **Compensation Service**.
2. Select the **Enable service at server startup** check box.
3. If required, modify the compensation handler retry interval and limit. These values control the frequency with which the compensation handler compensate and close methods are retried, when either throw a RetryCompensationHandlerException exception, and the number of times that these methods are retried.
4. Save your changes to the master configuration.
5. Repeat the previous steps for each server that you plan to use.
6. Restart all the servers for the changes to take effect.

Results

The business activity support is enabled for the application server. Verify a successful enablement by checking for the message, `CWSCP0005I: The Compensation service started successfully.` in the `SystemOut.log` file for the relevant server.

What to do next

Deploy the business-activity-enabled application to the server.

Creating an application that uses the Web Services Business Activity support

To create an application component that uses the business activity support, you must set **Run EJB methods under a Business Activity scope** in the deployment descriptor of the relevant application component, and if required, create and specify a compensation handler for the application to use if there is an error. You then build the component into the application and deploy the application onto a server that has the business activity support enabled. The application component can be either an enterprise bean or a Web service that is implemented as an enterprise bean.

Before you begin

For information about editing deployment descriptors, refer to topics about the Application Deployment Descriptor editor in the Application Server Toolkit information, in the navigation pane of this information center.

About this task

Perform this task for an application that runs on a business-activity-enabled sever to use the business activity support at run time, and to perform work that might later be compensated by a compensation handler. If the application requires compensation when a business activity scope ends, the application passes the data that is required by the compensation process to a compensation handler indirectly, by using the business activity API. The data that is required by the compensation process can be in the form of either a serializable object or a Service Data Object (SDO).

1. Design the application component that requires the business activity support. In particular, define the application component requirements for compensation and close activities. If the application component requires compensation, define the nature of the data in the serializable object or the SDO that the application component passes to the compensation handler.
2. Using the information from your application design, create the compensation handler for the application component, if required. This handler defines the close and compensation logic that runs upon completion of a business activity scope that has the handler added to it through an application component.
   a. Open your chosen WebSphere Application Server assembly tool.
   b. Create a new Java class that implements the appropriate interface, depending on the format of the data that is required by the compensation process:
      - For a serializable object, implement the com.ibm.websphere.wsba.serializable.CompensationHandler interface.
      - For an SDO, implement the com.ibm.websphere.wsba.CompensationHandler interface.
   c. Implement the close and compensate methods on the new compensation handler object, to perform appropriate actions depending on the serializable or SDO data that passes to the handler when it is invoked.

   The compensation handler class is now ready for the application component to reference, and for assembly into an application.
3. Open the application component in the assembly tool.
4. Open the deployment descriptor for the application component in the deployment descriptor viewer.
5. Scroll to the **Compensation** section and select the **Run EJB methods under a Business Activity scope** check box.
6. In the **Compensation handler class** text field, type the fully qualified class name of the compensation handler class that you created earlier.
7. Save the deployment descriptor.

8. Build the application, including both the application component and the compensation handler. If the application is a Web service, the application must be compliant with the Java Specification Request (JSR) 109 standard.
9. Deploy the application onto an application server that is business-activity-enabled.

Results

The application is now business-activity-enabled, and can use the business activity support at run time through the business activity API. The application component has a compensation handler associated with it, and can therefore call the setCompensationDataImmediate and setCompensationDataAtCommit methods at run time to add the compensation handler to the business activity scope. For more information about these methods, see “Business activity API” on page 702. If the unit of work with which the business activity scope is associated fails, the compensation handler performs actions to compensate for the error.

What to do next

Ensure that the compensation handler class is on the application class path for the WebSphere Application Server runtime environment.

Business activity API

Use the business activity application programming interface (API) to create business activities and compensation handlers for an application component, and to log data that is required to compensate an activity if there is a failure in the overall business activity.

Overview

The business activity support provides a UserBusinessActivity API and two interfaces: a serializable.CompensationHandler interface and a CompensationHandler interface. Each interface has two exceptions: RetryCompensationHandlerException and CompensationHandlerFailedException. You can look up the UserBusinessActivity interface from the application server Java Naming and Directory Interface (JNDI) at `java:comp/websphere/UserBusinessActivity`. For example:

```
InitialContext ctx = new InitialContext();
UserBusinessActivity uba = (UserBusinessActivity) ctx.lookup("java:comp/websphere/UserBusinessActivity");
```

You can use the getId method to access the unique identifier for the business activity that is currently associated with the calling thread. The identifier is the same as the one that is generated for the business activity scope at run time and that is used for information, warning, and error messages. For example, the application can use the identifier in audit or diagnostic messages, and it is possible to correlate between application-generated and runtime-generated messages.

```
InitialContext initialContext = new InitialContext();
UserBusinessActivity uba = initialContext.lookup("java:comp/websphere/UserBusinessActivity");
...
String activityId = uba.getId();
if (activityId == null)
// No activity on the thread
else
// Output audit message including activity id
```

If an application component runs work that might require compensating upon failure in the business activity, you must provide a compensation handler class that is assembled as part of the deployed application. This Java class must implement one of the following interfaces:

- com.ibm.websphere.wsba.serializable.CompensationHandler, which takes a takes a parameter of a serializable object
- com.ibm.websphere.wsba.CompensationHandler, which takes a parameter of a Service Data Object (SDO)

Typically, applications that already have their data available in DataObject format will use the CompensationHandler interface, and applications that do not will use the serializable.CompensationHandler interface. Both interfaces support the close and compensate methods.

An application must register a compensation handler implementation that works with the type of compensation data (serializable object or SDO) that the application uses. If there is a mismatch between the type of data that the application component uses and the compensation handler implementation, there is an error.

During normal application processing, the application can make one or more invocations to the `setCompensationDataImmediate` or `setCompensationDataAtCommit` methods, passing in either a serializable object or an SDO that represents the current state of the work performed.

When the underlying unit of work (UOW) that the root business activity is associated with completes, all registered compensators are coordinated to complete. During completion, either the compensate or the close method is called on the compensation handler, passing in the most recent compensation data logged by the application component as a parameter. Your compensation handler implementation must be able to understand the data that is stored in either the serializable object or the SDO DataObject; using this data, the compensation handler must be able to determine the nature of the work performed by the enterprise bean and compensate or close in an appropriate way, for example by undoing changes made to database rows if there is a failure in the business activity. You associate the compensation handler with an application component by using the assembly tooling, such as Rational Application Developer.

Active and inactive compensation handlers

You implement the serializable.CompensationHandler or CompensationHandler interface for any application component that executes code that might need to be compensated within a business activity scope. Compensation handler objects are registered implicitly with the business activity scope under which the application runs, whenever the application calls the UserBusinessActivity API to specify compensation data. Compensation handlers can be in one of two states, active or inactive, depending on any transactional UOW under which they are registered. A compensation handler that is registered within a transactional UOW might initially be inactive until the transaction commits, at which point the compensation handler becomes active (see the following section). A compensation handler that is registered outside a transactional UOW always becomes active immediately.

When a business activity completes, it drives only active compensation handlers. Any inactive compensation handlers that are associated with the business activity are discarded and never driven.

Logging compensation data

The business activity API specifies two methods that allow the application to log compensation data. This data is made available to the compensation handlers during their processing when the business activity completes. The application calls one of these methods, depending on whether it expects transactions to be part of the business activity.

`setCompensationDataAtCommit()`

Call the setCompensationDataAtCommit method when the application expects a global transaction on the thread.

- If a global transaction is present on the thread, the CompensationHandler object is initially inactive. If the global transaction fails, it rolls back any transactional work done within its transaction context in an atomic manner, and drives the business activity to compensate other completed UOWs. The compensation handler does not need to be involved. If the global transaction commits successfully, the compensation handler becomes active because if the overall business activity fails, the compensation handler is required to compensate the durable work that is completed by the global transaction. The setCompensationDataAtCommit method configures the CompensationHandler instance to perform this compensation function.

- If a global transaction is not present when the setCompensationDataAtCommit method is called, the compensation handler becomes active immediately.

For example, for an SDO, using the same business activity instance as in the previous example:

```
DataObject compensationData = doWorkWhichWouldNeedCompensating();
uba.setCompensationDataAtCommit(compensationData);
```

setCompensationDataImmediate()

Call the setCompensationDataImmediate method when the application does not expect a global transaction on the thread.

The setCompensationDataImmediate method makes a CompensationHandler instance active immediately, regardless of the current UOW context at the time that the method is invoked. The compensation handler is always able to participate during completion of the business activity.

The role of the setCompensationDataImmediate method is to compensate any non-transactional work, in other words, work that can be performed either inside or outside a global transaction, but that is not governed by the transaction. An example of this type of work is sending an e-mail. The compensation handler must be active immediately so that if a failure occurs in a business activity, this non-transactional work is always compensated.

For example, for a serializable object, using the same business activity instance as in the previous example:

```
Serializable compensationData = new MyCompensationData();
uba.setCompensationDataImmediate(compensationData);
```

Although these two compensation data logging methods, if called in the same enterprise bean, use the same compensation handler class, they create two separate instances of the compensation handler class at run time. Therefore, the actions of the methods are mutually exclusive; calling one of the methods does not overwrite any work carried out by the other method.

If a compensation handler instance is already added to the Business Activity using one of these methods, and then the same method is called, passing in `null` as a parameter, that compensation handler instance is removed from the business activity, and is not driven to close or compensate during completion of the business activity.

As described previously, the business activity support adds a compensation handler instance to the business activity when a compensation data logging method is called for the first time by the enterprise bean that uses that business activity. At the same time, a snapshot of the enterprise application context is taken and logged with the compensation data. When the business activity competes, all the compensation handlers that were added to the business activity are driven to compensate or close. The code that you create in the CompensationHandler or serializable.CompensationHandler class is guaranteed to run in the same enterprise application context that was captured in the earlier snapshot.

For details about the methods available in the business activity API, see Additional Application Programming Interfaces (APIs).

Using WS-Policy to exchange policies in a standard format

WS-Policy is an interoperability standard that is used to describe and communicate the policies of a Web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction. WebSphere Application Server conforms to the WS-Policy specification, so that policy information can be exchanged and received in accordance with the WS-Policy standard.

About this task

For more information about using WS-Policy, see the following topics.

- Learn about WS-Policy
- Configure a service provider to share its policy configuration
- Configure the client policy using a service provider policy
- Configure a service provider to share its policy configuration using the wsadmin tool
- Configure the client policy based on a service provider policy using the wsadmin tool
- Configure security for a WS-MetadataExchange request

Learning about WS-Policy

WS-Policy is an interoperability standard that is used to describe and communicate the policies of a Web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction.

WebSphere Application Server conforms to the Web services Policy Framework (WS-Policy) specification. You can use the WS-Policy protocol to exchange policies in standard format. A policy represents the capabilities and requirements of a Web service, for example whether a message is secure and how to secure it, and whether a message is delivered reliably and how this is achieved. You can communicate the policy configuration to any other client, service registry, or service that supports the WS-Policy specification, including non-WebSphere Application Server products in a heterogeneous environment.

For a service provider, the policy configuration can be shared in published Web Services Description Language (WSDL), WSDL that is obtained by a client using an HTTP Get request, or by using the Web Services Metadata Exchange (WS-MetadataExchange) protocol. The WSDL is in the standard WS-PolicyAttachments format.

For a client, the client can obtain the policy of the service provider in the standard WS-PolicyAttachments format and use this information to establish a configuration that is acceptable to both the client and the service provider. In other words, the client can be configured dynamically, based on the policies supported by its service provider. The provider policy can be attached at the application or service level.

The WS-Policy assertion specifications that are supported in this version of WebSphere Application server are:

- WS-Policy. See Web Services Policy 1.5
- WS-Addressing. See Web Services Addressing 1.0 - Metadata.
- WS-AtomicTransaction. See Web Services Atomic Transaction Version 1.0 and Web Services Atomic Transaction Version 1.1.
- WS-ReliableMessaging. See Web Services Reliable Messaging Policy Assertion Version 1.0 and Web Services Reliable Messaging Policy Assertion Version 1.1.
- WS-SecurityPolicy. See WS-SecurityPolicy 1.2.

For details of the WS-Policy domains that are supported, see the following topics:

- “WS-Addressing policy settings” on page 911
- “WS-ReliableMessaging settings” on page 908
- “WS-Security policy settings” on page 935
- “WS-Transaction policy settings” on page 932

Web service providers and policy configuration sharing

A WebSphere Application Server service provider can share its current policy configuration through its Web Service Description Language (WSDL). The policy configuration is in standard WSDL

WS-PolicyAttachment format so that it can be shared with other clients, service registries, or services that support the Web Services Policy (WS-Policy) specification.

You can make the policy configuration of a Java API for XML-Based Web Services (JAX-WS) service endpoint available to share in two ways:

- Include the policy configuration of the service provider in the WSDL. The WSDL is then available to publish, or to obtain using an HTTP Get request.
- Enable the Web Services Metadata Exchange (WS-MetadataExchange) protocol so that the policy configuration of the service provider is included in the WSDL and is available to a WS-MetadataExchange GetMetadata request. An advantage of using the WS-MetadataExchange protocol is that you can apply message-level security to WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

The policy configuration in the WSDL is in the standard WS-PolicyAttachments format. Any WS-Policy attachments that were in the WSDL previously are removed. Note that policy configuration information becomes available in the WSDL to publish, but it is not available if you just view the WSDL document using the administrative console. Also, policy configuration information is not available in WSDL that is published remotely by using an administrative agent.

By default, the policy configuration of the service provider is not included in the WSDL. To include the policy configuration of the service provider in the WSDL, and specify how it is shared, you can use the administrative console or wsadmin commands.

Application developers can specify that a service provider shares its policy configuration, and how it is shared, using Rational Application Developer tools when a Web service is generated. For more information, see the Rational Application Developer documentation.

Transport policy information is not included in the policy configuration because transport policies such as HTTP, SSL, and JMS cannot be expressed in WS-PolicyAttachment format.

Bootstrap policy information, for example, the policy to access a WS-Trust service, can be included in the policy configuration if the bootstrap policy is expressed in WS-PolicyAttachment format.

You can configure a service provider to share its policy configuration at application or service level. The policy configuration that is represented by the policy sets attached to any lower levels will also be shared. Policy sets that are attached at lower levels override the policy set configuration attached at a higher level.

Troubleshooting policy configuration sharing

A service provider might not be able to share its policy configuration because the configuration cannot be expressed in the standard WS-PolicyAttachments format. One reason might be because multiple incompatible policies are defined for a particular attach point. Another reason might be because there is not enough binding information to generate the standard policy. Policy configuration might include bootstrap policy, for example, the policy to access a WS-Trust service, so the bootstrap policy must also be expressed in WS-PolicyAttachments format.

If the policy configuration cannot be shared, an error that describes the problem is written to the service provider error log, and the following policy is attached to the WSDL of the service provider:

```
<wsp:Policy>
<wsp:ExactlyOne>
</wsp:ExactlyOne>
</wsp:Policy>
```

This policy notifies the client that there is no acceptable policy configuration for the service. Other aspects of the WSDL are unaffected.

Web service clients and policy configuration using the service provider policy

If a service provider publishes its policy in its Web Services Description Language (WSDL), the policy configuration of a WebSphere Application Server service client can be configured dynamically, based on the policies supported by its service provider.

The service provider must publish its policy in WS-PolicyAttachment format in its WSDL and the client must be able to support those provider policies. The client can base its policy configuration entirely on the policy of the provider, or partly on the policy of the provider with restrictions that are defined by the policy set configuration of the client.

A client acquires the provider policy by using either an HTTP Get request or the Web Services Metadata Exchange (WS-MetadataExchange) protocol to obtain the WSDL of the provider. You can configure how the client obtains the provider policy, and the endpoint at which the policy is acquired, by using the administrative console or wsadmin commands. If you use the WS-MetadataExhange protocol to obtain the policy of the provider, this has the advantage that you can secure WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

The Web application client-side policy is calculated and cached as a runtime configuration. This calculated policy is known as the effective policy and is used for subsequent outbound Web service requests to the endpoint or operation for which the calculation was performed. The original policy set configuration of the client does not change.

For a specific set of WSDL, dynamic policy configuration occurs once, and it is assumed that this configuration is the same for all endpoints that implement a service, because they have the same WSDL. The policy calculations that are based on this WSDL are cached in the client runtime (they are not persisted) and shared with each target service.

If you require a different policy configuration for each endpoint implementation, you must create a new port for each endpoint. Then you can specify a different policy configuration for each endpoint.

Transport policies such as HTTP, SSL, and JMS, cannot be expressed in WS-PolicyAttachment format, so the client cannot acquire the transport policies of the service provider. If the client requires transport policies, you must configure these policies as part of the policy set configuration of the client.

Policy inheritance

The provider policy can be attached at the application or service level. Endpoints and operations inherit their policy configuration from the relevant service.

Calculating policy

Policy intersection is the comparison of a client policy and a provider policy to determine whether they are compatible, and the calculation of a new policy that complies with both their requirements and capabilities. When you obtain the policy of a service provider, you can choose to use the provider policy only, or to use the client and the provider policy. The outcome of policy intersection is as follows:

- When you specify provider policy only, the calculated policy is based on all the policies that the WebSphere Application Server client supports intersected by the provider policy. Effectively, the provider determines the policy, as long as the client can support that policy. This policy configuration is available if the scope point (endpoint operation) where the provider policy is attached is not attached to a client policy set and does not inherit a policy set attachment from parent scope points.
- When you specify client and provider policy, the calculated policy is based on the policy that is acceptable to the client intersected by the provider policy. Effectively, the policy conforms to the client policy set, but might be restricted further by the policies dictated by the provider. The policy that is acceptable to the client is defined by the policy set that is either attached to the client scope point, or that the client scope point inherits from a parent scope point. This policy configuration is available if the

scope point (endpoint operation) where the provider policy is attached is attached to a client policy set or inherits a policy set attachment from parent scope points.

The WS-Policy language provides a way to express multiple policy choices, so the policy calculation might produce more than one result. For example, the service provider might support both WS-ReliableMessaging 1.0 and WS-ReliableMessaging 1.1. If the client also supports both versions, the client could use either version in its Web service requests to the provider. In this situation, where more than one specification version is acceptable to both the client and the provider, the effective policy is calculated using the most recent version.

Policy intersection in the JAX-WS dispatch client

Invocations that use the JAX-WS dispatch client (javax.xml.ws.Dispatch) use provider policy in their configuration if this is the administered behavior for the service. If the operation for the invocation is unknown, the client behaves as follows:

- The client complies with the provider policy scoped to the operation only if the provider policy is identical for all the operations provided by the service (both semantically and syntactically).
- If the provider policy is not identical for all the operations provided by the service, the client returns a JAX-WS WebserviceException with the cause WSPolicyException (CWPOL0106E), and an appropriate error message.
- If there is no policy on any of the operations, the client uses the effective provider policy for the service endpoint.

Refreshing the provider policy held by the client

The provider policy that the client holds for a service is refreshed the first time that the Web service is invoked after the application is loaded. After that, the provider policy is refreshed when the application restarts, or when you explicitly invoke an update of the provider policy.

You can invoke an update of the provider policy in the application code. This might be useful if a JAX-WS invocation fails; in the exception handling, you can force a retry with refreshed policy. You can set the following property (available in the WSPConstants class of the API) on the JAX-WS client proxy, then reissue the JAX-WS request: com.ibm.wsspi.wspolicy.refreshProviderPolicy.

When the com.ibm.wsspi.wspolicy.refreshProviderPolicy property is set, the provider policy that the client holds for a service is refreshed, and the effective policy is recalculated at the next request. After the refresh and recalculation have occurred, the com.ibm.wsspi.wspolicy.refreshProviderPolicy property is unset.

After an update of the policy provider is invoked, the next time that the Web service that is associated with the endpoint you selected is invoked, the policy of the service provider is obtained and the policy intersection is recalculated.

The following example of code for a dispatch client shows the identification of an exception that might be resolved by refreshing the provider policy, followed by the invocation of the refresh.

```
try
{
dispatch.invoke(params);
}
catch (javax.xml.ws.WebServiceException e)
{
Throwable cause = e.getCause();
if ((cause instanceof NullPolicyException) || (cause instanceof PolicyException) )
{
// The exception might be because the policy of the provider is not up to date.
//
// There is also a message on the console that starts with the characters CWPOL,
// which helps to decipher and debug the cause of the error.
// This message is also available by using
```

```
// String nlsedMessage = cause.getMessage();
Map<String, Object> requestContext = dispatch.getRequestContext();
requestContext.put(WSPConstants.REFRESH_PROVIDER_POLICY, Boolean.TRUE);
// The following method might cause another jax-ws invocation exception.
// The cause might still be policy, in which case, a message is written to the
// console.
dispatch.invoke(params);
}
// For all other exceptions, use the normal exception handling for the
// application. In this case, assume there are no other exceptions and rethrow the
// initial exception. Remember that the WebServiceException might be caused by a
// WSPolicyAdministrationException. In this situation, a message is written to the
// console, but forcing a refresh in the application cannot resolve the problem.
throw e;
}
```

WS-MetadataExchange requests

You can use the Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request to exchange Web Services Definition Language (WSDL) that is annotated with WS-Policy information. A service provider can use a WS-MetadataExchange request to share its policies, and a service client can use a WS-MetadataExchange request to apply the policies of a provider. You can secure WS-MetadataExchange requests by using transport-level or message-level security.

The WS-MetadataExchange specification defines a mechanism to retrieve metadata from an endpoint. WebSphere Application Server supports the use of the WS-MetadataExchange 1.1 GetMetadata request to return metadata in a response. A service provider can use this mechanism to make WSDL that is annotated with WS-Policy information available, that is, the service provider can share its policies. A service client can use this mechanism to obtain WSDL that is annotated with WS-Policy information from a service provider and then apply those policies. The policy configuration must be in WS-PolicyAttachments format in the WSDL of the service provider.

You can use a WS-MetadataExchange request as an alternative to using an HTTP Get request.

By default, a service provider or a service client does not use WS-MetadataExchange to share or obtain WS-Policy information. You must configure the service provider to share its policies, or configure the service client to apply the policies of a service provider, and specify that a WS-MetadataExchange request is used to share or obtain the policy configuration. WS-Policy information can be shared or obtained at the application or service level. You can configure the service provider or service client by using the administrative console or using wsadmin commands.

Application developers can configure the service provider or service client using Rational Application Developer tools when a Web service is generated. For more information, see the Rational Application Developer documentation.

When a service provider is configured to share its policies through WS-MetadataExchange, the service supports incoming WS-MetadataExchange GetMetadata requests that are limited to the WSDL dialect. When the service receives such a request, the WSDL of the service is returned inline through a conformant WS-MetadataExchange response. The WSDL of the service contains WS-PolicyAttachments annotations that represent the current policy configuration. The policy configuration is in WS-PolicyAttachments format in the WSDL so that it is then available to other clients, service registries or services that support the Web Services Policy (WS-Policy) specification and the WS-MetadataExchange GetMetadata request.

When a service client is configured to use WS-MetadataExchange to obtain the policy of a service provider, the service client sends a WS-MetadataExchange GetMetadata request that specifies the WSDL dialect whenever it needs to obtain or refresh the policy of the provider.

WS-MetadataExchange security

You must ensure that the GetMetadata request is secured so that there is effective authentication, authorization, integrity, and confidentiality. End-to-end authentication is particularly important for the exchange of security metadata (SecurityPolicy), because if an unauthorized party could access this information, security credentials could be sent to non-trusted endpoints.

The GetMetadata request is targeted at the same port as the application endpoint, so if the application uses transport-level security, the GetMetadata request is also be targeted at the secure port and will, by default, use the same transport-level security configuration of the application.

Additionally, you can apply message-level security (WS-Security) to the metadata exchange. You might want to apply message-level security if transport-level security is not available on the application endpoint, or if transport-level security is not adequate for your requirements. An advantage of message-level security is that it provides end-to-end security by incorporating security features in the header of the SOAP message.

To provide message-level security, you attach system policy sets and general (named) bindings to the endpoint when you configure the service provider or service client to exchange policy configurations.

System policy sets are used for system messages that are not business-related, whereas application policy sets specify policy assertions for business-related messages. For example, system policy sets are used for messages that apply qualities of service (QoS), which includes the messages that are defined in the WS-MetadataExchange protocol. To provide message-level security for a GetMetadata request, you must attach a system policy set that contains only Web Services Security (WS-Security) or Web Services Addressing (WS-Addressing) policies. You can specify general bindings that are scoped either to the global domain or to the security domain of the service.

When you apply message-level security, any transport policy of the application is always used.

Configuring a service provider to share its policy configuration

A WebSphere Application Server service provider can share its policy configuration in published Web Services Description Language (WSDL), or WSDL that is obtained using an HTTP Get request or the Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request.

Before you begin

You have developed a Web services service provider that contains all the necessary artifacts and deployed your Web services application into your application server instance. You have attached the policy sets and managed the associated bindings.

For a list of WS-Policy assertion specifications and WS-Policy domains that are supported, see “Learning about WS-Policy” on page 715.

About this task

You can make the policy configuration of a Java API for XML-Based Web Services (JAX-WS) service endpoint available to share in two ways:

- Include the policy configuration of the service provider in the WSDL. The WSDL is then available to publish, or to obtain using an HTTP Get request.
- Enable the Web Services Metadata Exchange (WS-MetadataExchange) protocol so that the policy configuration of the service provider is included in the WSDL and is available to a WS-MetadataExchange GetMetadata request. An advantage of using the WS-MetadataExchange protocol is that you can apply message-level security to WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

The policy configuration in the WSDL is in the standard WS-PolicyAttachments format. Any WS-Policy attachments that were in the WSDL previously are removed. Note that policy configuration information becomes available in the WSDL to publish, but it is not available if you just view the WSDL document using the administrative console. Also, policy configuration information is not available in WSDL that is published remotely by using an administrative agent.

You must configure a service provider to share its policy configuration because by default the policy configuration is not available in its WSDL. You can configure the service provider to include the policy configuration in its WSDL, to use WS-MetadataExchange so that the policy configuration is available, or both. This topic describes how to configure a service provider to share its policy configuration using the administrative console. You can also configure a service provider to share its policy configuration by using wsadmin commands or Rational Application Developer tools.

You can configure a service provider to share its policy configuration at application or service level. The policy configuration that is represented by the policy sets attached to any lower levels will also be shared. Policy sets that are attached at lower levels override the policy set configuration attached at a higher level.

1. From the navigation pane of the administrative console, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_provider_application_instance*** → **[Web services properties] Service provider policy sets and bindings**.
2. In the row for the application or service where the provider policy that you want to share is attached, click the link in the Policy sharing column. The link is either **Enabled** or **Disabled**. The Policy Sharing pane is displayed.
3. To include the policy configuration of the service provider in its WSDL so that it can be either published or obtained using an HTTP Get request, select **Exported WSDL**.
4. To enable WS-MetadataExchange and make the policy configuration of the service provider available to a WS-MetadataExchange GetMetada request, select **WS-MetadataExchange request**.
5. Optional: If you select **WS-MetadataExchange request** and you want to use message-level security, select **Attach a system policy set to the WS-MetadataExchange**, then select a suitable policy set and binding from the drop-down lists. See “Configuring security for a WS-MetadataExchange request” on page 729.
6. Click **OK** and save your changes to the master configuration.

Results

The policy configuration of the service provider is available to its clients. The WSDL of the service provider contains the current policy configuration in WS-PolicyAttachments format so that it is available to other clients, service registries, or services that support the Web Services Policy (WS-Policy) specification. The link in the Policy Sharing column on the Service provider policy sets and bindings pane changes to **Enabled**.

If the policy configuration cannot be shared, an error that describes the problem is written to the service provider error log, and the following policy is attached to the WSDL of the service provider:

```
<wsp:Policy>
<wsp:ExactlyOne>
</wsp:ExactlyOne>
</wsp:Policy>
```

This policy notifies the client that there is no acceptable policy configuration for the service. Other aspects of the WSDL are unaffected.

A service provider might not be able to share its policy configuration because the configuration cannot be expressed in the standard WS-PolicyAttachments format. One reason might be because multiple incompatible policies are defined for a particular attach point. Another reason might be because there is not enough binding information to generate the standard policy. Policy configuration might include

bootstrap policy, for example, the policy to access a WS-Trust service, so the bootstrap policy must also be expressed in WS-PolicyAttachments format.

Configuring a service provider to share its policy configuration using the wsadmin tool

A WebSphere Application Server service provider can share its policy configuration in published Web Services Description Language (WSDL), or WSDL that is obtained using an HTTP Get request or the Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request.

Before you begin

You have developed a Web services service provider that contains all the necessary artifacts and deployed your Web services application into your application server instance. You have attached the policy sets and managed the associated bindings.

For a list of WS-Policy assertion specifications and WS-Policy domains that are supported, see “Learning about WS-Policy” on page 715.

About this task

You can make the policy configuration of a Java API for XML-Based Web Services (JAX-WS) service endpoint available to share in two ways:

- Include the policy configuration of the service provider in the WSDL. The WSDL is then available to publish, or to obtain using an HTTP Get request.
- Enable the Web Services Metadata Exchange (WS-MetadataExchange) protocol so that the policy configuration of the service provider is included in the WSDL and is available to a WS-MetadataExchange GetMetadata request. An advantage of using the WS-MetadataExchange protocol is that you can apply message-level security to WS-MetadataExchange GetMetadata requests by using a suitable system policy set.

The policy configuration in the WSDL is in the standard WS-PolicyAttachments format. Any WS-Policy attachments that were in the WSDL previously are removed. Note that policy configuration information becomes available in the WSDL to publish, but it is not available if you just view the WSDL document using the administrative console. Also, policy configuration information is not available in WSDL that is published remotely by using an administrative agent.

You must configure a service provider to share its policy configuration because by default the policy configuration is not available in its WSDL. You can configure the service provider to include the policy configuration in its WSDL, to use WS-MetadataExchange so that the policy configuration is available, or both. This topic describes how to configure a service provider to share its policy configuration using wsadmin commands. You can also use the administrative console or Rational Application Developer (RAD) tools.

You can configure a service provider to share its policy configuration at application or service level. The policy configuration that is represented by the policy sets attached to any lower levels will also be shared. Policy sets that are attached at lower levels override the policy set configuration attached at a higher level.

1. Start the wsadmin tool if it is not already running.
2. Use the SetProviderPolicySharingInfo command. For example:

```
AdminTask.setProviderPolicySharingInfo('[-applicationName WebServiceProviderApplication
-resource WebService:/WebServiceProvider.war:{http://example_path/}Service1
-sharePolicyMethods [httpGet ]]')
```

3. Save your changes to the master configuration.

   To save your configuration changes, enter the following command:

```
AdminConfig.save()
```

Results

The policy configuration of the service provider is available to its clients. The WSDL of the service provider contains the current policy configuration in WS-PolicyAttachments format so that it is available to other clients, service registries, or services that support the Web Services Policy (WS-Policy) specification.

If the policy configuration cannot be shared, an error that describes the problem is written to the service provider error log, and the following policy is attached to the WSDL of the service provider:

```
<wsp:Policy>
<wsp:ExactlyOne>
</wsp:ExactlyOne>
</wsp:Policy>
```

This policy notifies the client that there is no acceptable policy configuration for the service. Other aspects of the WSDL are unaffected.

A service provider might not be able to share its policy configuration because the configuration cannot be expressed in the standard WS-PolicyAttachments format. One reason might be because multiple incompatible policies are defined for a particular attach point. Another reason might be because there is not enough binding information to generate the standard policy. Policy configuration might include bootstrap policy, for example, the policy to access a WS-Trust service, so the bootstrap policy must also be expressed in WS-PolicyAttachments format.

What to do next

Optionally, you can publish the WSDL files.

Policy sharing settings

Use this pane to view and change whether the policy configuration of a Web services service provider is shared. You can configure the service provider to include the policy configuration in its Web Service Description Language (WSDL) so that it can be accessed using an HTTP Get request, or published. You can also make the policy configuration available to a Web Services Metadata Exchange (WS-MetadataExchange) request.

To view this administrative console page for an application or service, complete the following steps. The application must be a Web services service provider.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_provider_application_name*** → **[Web Services Properties] Service provider policy sets and bindings**.
2. Select the link in the Policy Sharing column for the application or service you require. The link is available if the application or service has a policy set attached.

To view this administrative console page for a service, complete the following steps.

1. Click **Services** → **Service providers** → ***service_name***.
2. Select the link in the Policy Sharing column for the service. The link is available if the service has a policy set attached.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Exported WSDL:

Select **Exported WSDL** to include the policy configuration of the service provider in the WSDL. The policy configuration is in WS-PolicyAttachments format in the WSDL so that it is then available to other clients, service registries, or services that support the Web Services Policy (WS-Policy) specification. The policy

configuration will be available in published WSDL, or a client can use an HTTP Get request that is targeted at the target URL followed by `?WSDL` to obtain the provider policy.

WS-MetadataExchange request:

Select **WS-MetadataExchange** to make the policy configuration of the service provider available to a WS-MetadataExchange GetMetadata request. The policy configuration is in WS-PolicyAttachments format in the WSDL so that it is then available to other clients, service registries or services that support the Web Services Policy (WS-Policy) specification and the WS-MetadataExchange GetMetadata request.

When **WS-MetadataExchange** is selected, the **Attach a system policy set to the WS-MetadataExchange** option is available.

Attach a system policy set to the WS-MetadataExchange:

Select **Attach a system policy set to the WS-MetadataExchange** to set message-level security for the WS-MetadataExchange request. By default, this option is not selected and the transport policy of the application is used. This option is available only when **WS-MetadataExchange** is selected.

When **Attach a system policy set to the WS-MetadataExchange** is selected, the **Policy set** and **Binding** lists are available.

Policy set:

Select the policy set you require from the list to provide message-level security for the WS-MetadataExchange request. You can select from system policy sets that contain only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault.

System policy sets are used for system messages that are not business-related, for example, messages that apply qualities of service (QoS), including the messages that are defined in the WS-MetadataExchange protocol.

Note that any transport policy of the application is always used.

This option is available only when **Attach a system policy set to the WS-MetadataExchange** is selected.

Binding:

Select the binding you require from the list to provide message-level security for the WS-MetadataExchange request. You can select from general bindings that are scoped to the global domain, or scoped to the security domain of this service.

This option is available only when **Attach a system policy set to the WS-MetadataExchange** is selected.

Configuring the client policy using a service provider policy

An application that is a Web service client can obtain the policy configuration of a Web service provider and use this information to establish a policy configuration that is acceptable to both the client and the service provider.

Before you begin

You have developed a Web service client that contains all the necessary artifacts, and deployed your Web services application into your application server instance. If you require them, you have attached the policy sets and managed the associated bindings.

The service provider must publish its policy in its Web Services Description Language (WSDL) and that policy must contain its policy configuration at run time in WS-PolicyAttachments format. The client must be able to support those provider policies.

For a list of WS-Policy assertion specifications and WS-Policy domains that are supported, see “Learning about WS-Policy” on page 715.

About this task

The client can obtain the policy of the service provider in the standard WS-PolicyAttachments format. You can apply the provider policy at the application or service level. Endpoints and operations inherit their policy configuration from the relevant service.

Policy intersection is the comparison of a client policy and a provider policy to determine whether they are compatible, and the calculation of a new policy, known as the effective policy, that complies with both their requirements and capabilities.

This topic describes how to configure the client policy to use a service provider policy by using the administrative console. You can also configure the client policy to use a service provider policy by using wsadmin commands.

1. From the navigation pane of the administrative console, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_client_application_instance*** → **[Web services properties] Service client policy sets and bindings**.
2. In the row for the application or service where you want to apply the policy, click the link in the Policies Applied column. The Policies Applied pane is displayed.
3. Select one of the following options from the drop-down list:
   - Provider policy only. Configure the client based solely on the policy of the service provider. This option is available when a client policy set is not attached.
   - Client and provider policy. Configure the client based on both the client policy set and the policy of the service provider. This option is available when a client policy set is attached.

   The other options in the list do not apply to this task.
4. To obtain the provider policy using an HTTP Get request, click **HTTP Get request**. By default, the HTTP Get request is targeted at the URL for the service endpoint followed by `?WSDL`. When you apply a policy to an application, you cannot change this value.
   a. Optional: If you are applying a policy to a service and you want to change the target of the HTTP Get request from the default, click **Specify request target**, then enter the URL you require, followed by `?WSDL`, in the field. For example:

```
http://service1/service1?WSDL
```

   b. Optional: If you are applying a policy to a service and you want to change the target of the HTTP Get request to the default, click **Use the default request target**. The HTTP Get request is targeted at the URL for the service endpoint followed by `?WSDL`.
5. To obtain the provider policy using a Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request, click **WS-MetadataExchange request**.
6. Optional: If you select **WS-MetadataExchange request** and you want to use message-level security, select **Attach a system policy set to the WS-MetadataExchange**, then select a suitable policy set and binding from the drop-down lists. See “Configuring security for a WS-MetadataExchange request” on page 729.
7. Click **OK**.
8. Save your changes to the master configuration.

Results

The Web application client-side policy is calculated based either on the policy of the service provider, or on the client policy set and the policy of the service provider, depending on which option you selected, and cached as a runtime configuration. The effective policy is used for subsequent outbound Web service requests to the endpoint or operation for which the dynamic policy calculation was performed. The policy set configuration of the client does not change.

The provider policy that the client holds for a service is refreshed the first time that the Web service is invoked after the application is loaded. After that, the provider policy is refreshed when the application restarts, or if the application explicitly invokes a refresh.

Configuring the client policy based on a service provider policy using the wsadmin tool

An application that is a Web service client can obtain the policy configuration of a Web service provider and use this information to establish a policy configuration that is acceptable to both the client and the service provider.

Before you begin

You have developed a Web service client that contains all the necessary artifacts, and deployed your Web services application into your application server instance. If you require them, you have attached the policy sets and managed the associated bindings.

The service provider must publish its policy in its Web Services Description Language (WSDL) and that policy must contain its policy configuration at run time in WS-PolicyAttachments format. The client must be able to support those provider policies.

For a list of WS-Policy assertion specifications and WS-Policy domains that are supported, see “Learning about WS-Policy” on page 715.

About this task

The client can obtain the policy of the service provider in the standard WS-PolicyAttachments format. You can apply the provider policy at the application or service level. Endpoints and operations inherit their policy configuration from the relevant service.

Policy intersection is the comparison of a client policy and a provider policy to determine whether they are compatible, and the calculation of a new policy, known as the effective policy, that complies with both their requirements and capabilities.

This topic describes how to configure the client policy to use a service provider policy by using the administrative console. You can also configure the client policy to use a service provider policy by using wsadmin commands.

1. Start the wsadmin tool if it is not already running.
2. Use the SetClientDynamicPolicyControl command. For example:

```
AdminTask.setClientDynamicPolicyControl('[-applicationName WebServiceClientApplication
-resource WebService:/ClientApplication.war:{http://example_path/}Service1
-acquireProviderPolicyMethod [httpGet ]
-httpGetProperties [httpGetTargetURI http://example_path?WSDL]]')
```

3. Save your changes to the master configuration.

   To save your configuration changes, enter the following command:

```
AdminConfig.save()
```

Results

The Web application client-side policy is calculated based either on the policy of the service provider, or on the client policy set and the policy of the service provider, depending on which option you selected, and cached as a runtime configuration. The effective policy is used for subsequent outbound Web service requests to the endpoint or operation for which the dynamic policy calculation was performed. The policy set configuration of the client does not change.

The provider policy that the client holds for a service is refreshed the first time that the Web service is invoked after the application is loaded. After that, the provider policy is refreshed when the application restarts, or if the application explicitly invokes a refresh.

Policies applied settings

Use this pane to view and change whether the policy configuration of a WebSphere Application Server service client is configured dynamically, based on the policies supported by its service provider. You can view or change how the client obtains the policy of the service provider; the client can use an HTTP Get request or a Web Services Metadata Exchange (WS-MetadataExchange) request. You can specify a policy set and binding to provide message-level security for WS-MetadataExchange.

To view this administrative console page for an application or service, complete the following steps. The application must be a Web services service client.

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_client_application_name*** → **[Web Services Properties] Service client policy sets and bindings**.
2. Select the link in the **Policies Applied** column for the application or service you require.

To view this administrative console page for a service, complete the following steps:

1. Click **Services** → **Service clients** → ***service_name***.
2. Select the link in the **Policies Applied** column for the service. The link is available if the service or the parent application has a policy set attached.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Apply the following policies:

Specifies whether the client policy is based on the policy of the service provider, and how that policy is used.

Select from the following options:

- Client policy only. Configure the client based solely on the client policy set. Do not use the policy of the service provider. This option is available when a client policy set is attached to the resource.
- Client and provider policy. Configure the client based on both the client policy set and the policy of the service provider. This option is available when a client policy set is attached to the resource.
- Inherit application attachment. Inherit the setting of the parent application. This option is available for a service when a client policy set is not attached to the service. If there is a policy set attached to the parent application, the inherited properties are displayed on this pane, but you cannot change them. If there is no policy set attached to the parent application, when you return to the Service clients policy sets and bindings pane, the Policies applied column shows a value of **None**.
- Provider policy only. Configure the client based solely on the policy of the service provider. This option is available when a client policy set is not attached to the resource.

HTTP GET request:

Click **HTTP GET request** to obtain the policy of the service provider by using an HTTP Get request. The policy configuration must be in WS-PolicyAttachments format in the WSDL of the service provider.

This option is available when **Apply the following policies** is set to Client and provider policy or Provider policy only.

By default, the HTTP Get request is targeted at the URL for each service endpoint followed by ?WSDL.

Use the default request target:

When you apply a policy to a service, click **Use the default request target** to target the HTTP Get request at the URL for each service endpoint followed by ?WSDL.

This option is available when **HTTP GET request** is selected and you apply a policy to a service.

When you apply a policy to an application, this option is not available.

Specify request target:

When you apply a policy to a service, click **Specify request target** to change the target of the HTTP Get request. Enter the URL you require, followed by `?WSDL`, in the field.

This option is available when **HTTP GET request** is selected and you apply a policy to a service.

When you apply a policy to an application, this option is not available.

WS-MetadataExchange request:

Click **WS-MetadataExchange** to obtain the policy of the service provider by using a WS-MetadataExchange GetMetadata request. The policy configuration must be in WS-PolicyAttachments format in the WSDL of the service provider.

This option is available when **Apply the following policies** is set to Client and provider policy or Provider policy only.

Attach a system policy set to the WS-MetadataExchange:

Select **Attach a system policy set to the WS-MetadataExchange** to set message-level security for the WS-MetadataExchange request. By default, this option is not selected and the transport policy of the application is used. This option is available when **WS-MetadataExchange** is selected.

When **Attach a system policy set to the WS-MetadataExchange** is selected, the **Policy set** and **Binding** lists are available. If you select **Attach a system policy set to the WS-MetadataExchange**, you must also select a policy set and a binding.

Policy set:

Select the policy set you require from the list to provide message-level security for the WS-MetadataExchange request. You can select from system policy sets that contain only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault.

System policy sets are used for system messages that are not business-related, for example, messages that apply qualities of service (QoS), including the messages that are defined in the WS-MetadataExchange protocol.

Note that any transport policy of the application is always used.

This option is available when **Attach a system policy set to the WS-MetadataExchange** is selected.

Binding:

Select the binding you require from the list to provide message level security for the WS-MetadataExchange request. You can select from general bindings that are scoped to the global domain or scoped to the security domain of this service.

This option is available when **Attach a system policy set to the WS-MetadataExchange** is selected.

Configuring security for a WS-MetadataExchange request

You can configure message-level security for a Web Services Metadata Exchange (WS-MetadataExchange) GetMetadata request by specifying a suitable policy set and binding. You do this when you configure a Web service provider to share its policies or a Web service client to obtain the policies of a service provider.

Before you begin

For a service provider, you have completed the procedure to configure a service provider to share its policy configuration, up to and including the step to enable WS-MetadataExchange.

For a service client, you have completed the procedure to configure the client policy using a service provider policy, up to and including the step to use WS-MetadataExchange.

About this task

By default, the WS-MetadataExchange GetMetadata request uses the transport-level security configuration of the application. You might want to apply message-level security if transport-level security is not available on the application endpoint, or if transport-level security is not adequate for your requirements. An advantage of message-level security is that it provides end-to-end security, which is especially important for the exchange of security metadata.

This topic describes how to configure security for a WS-MetadataExchange request using the administrative console. You can also configure security for a WS-MetadataExchange request using wsadmin commands.

1. For a service provider, in the Policy Sharing panel on the administrative console, select **Attach a system policy set to the WS-MetadataExchange**. For a service client, in the Policies Applied panel on the administrative console, select **Attach a system policy set to the WS-MetadataExchange**.
2. Select a system policy set to provide message-level security from the Policy set list. You can select from system policy sets that contain only WS-Security policies, only WS-Addressing policies, or both. The default policy set is SystemWSSecurityDefault. If the policy sets that are listed are not suitable for your requirements, create your own system policy set, then return to this procedure.
3. Select a general binding for the policy set attachment from the Binding list. You can select from general bindings that are scoped to the global domain, or the security domain of this service. If the bindings that are listed are not suitable for your requirements, create your own general binding, then return to this procedure.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

Message-level security is applied to the WS-MetadataExchange GetMetadata request.

Administering deployed Web services applications

You can administer deployed Web services applications using the administrative console.

Before you begin

Before you can administer a Web service application, you need to deploy your Web service application.

About this task

You can use the administrative console to administer Java API for XML-Based Web Services (JAX-WS) service provider or service client applications or Java API for XML-based RPC (JAX-RPC) Web services.

- Administer service providers. You can administer your service providers using the following ways:
  - View service providers at the cell level using the administrative console. You can view the details of your service provider, manage the policy sets for the service, its endpoints and operations, and assign bindings for the policy set attachment at the cell level.
  - View service providers at the application level using the administrative console. You can view the details of your service provider, manage the policy sets for the service, its endpoints and operations, and assign bindings for the policy set attachment at the application level.
  - Manage policy sets and bindings for service providers. You can view the details of your service provider, manage the policy sets for the service, its endpoints and operations, and assign bindings for the policy set attachment.
  - Manage policy sets and bindings for service providers at the application level using the administrative console. You can manage policy sets for the provider, its endpoints, and operations, and assign bindings for the policy set attachment at the application level.
  - View WSDL document using the administrative console . You can view the WSDL document for your JAX-WS application.
- Administer service clients. You can administer your service clients using the following ways:
  - View service clients at the cell level . Your application server instance can have one or more applications deployed on it that contain service clients. You can view a list of your service clients at the cell level using the administrative console.
  - View service clients at the application level . Your application server instance can have one or more applications deployed on it that contains service clients. You can view the service client names that are referenced in an application.
  - Manage policy sets and bindings for service clients. You can view details of your service client reference, manage the policy sets for the service, its endpoints and operations, and assign bindings for the policy set attachment.
  - Manage policy sets and bindings for service clients at the application level. You can manage policy sets for your service client applications or its service references, endpoints, or operations and assign bindings for the policy set attachment at the application level.
- View the deployment descriptors.. View the Web services server and client deployment descriptors for a deployed Web services application. You can view the bindings in the deployment descriptors. The deployment descriptors are required for JAX-RPC Web services. You can optionally use the webservices.xml deployment descriptor to augment or override application metadata specified in annotations within your JAX-WS Web services.
- Configure the scope of a Web service port..**(JAX-RPC applications only)** When a Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise bean module, including implementation scope and client bindings information. There are three levels of scope that you can set: application, session and request.

Overview of service and endpoint listeners

The administration function of the product is enhanced to support the service listener. This topics describes the service listener and endpoint listener.

The administrative enhancements to the product include the ability to control throughput by starting and stopping individual service and endpoint listeners. By stopping a service listener, it effectively throttles back the inbound requests to all the endpoints for that service, while the whole application and the other services run to serve requests.

Service listener

The service listener can be started or stopped. When a service listener is stopped, all the endpoints for the service are stopped and new inbound requests are no longer processed. The endpoints for the service send out 404 error message response code to indicate that the service listener is currently stopped and cannot service new inbound requests. When the service listener is started, all the endpoints for the service resume processing of new inbound requests.

Endpoint listener

The endpoint listener can be started or stopped. When a service endpoint listener is stopped, the specific endpoint stops listening to any new inbound request. The requests that are already being processed are not affected. The endpoint sends out a 404 error message response code to indicate that the endpoint is currently stopped and cannot service a new, inbound request. When the endpoint listener is started, the specific endpoint resumes listening on new inbound requests. The state of a particular service endpoint does not impact the listening function of other endpoints for the same service.

Administration of service and endpoint listeners

The administration function of the product is enhanced to support service and endpoint listeners. MBeans such as EndpointManager and the EndpointCentralManager can be used to invoke service and endpoint listeners.

After an application containing a Web service is installed, you want to verify that the service is installed correctly. You also want to monitor its service listener state, and update the listener state as needed to control the throughput. One option is to use the collection view of service providers in the administrative console of the product to locate the service provider of interest and observe its listener state.

If you do not want the listener state, then select the service and choose to start or stop the service listener. As the system starts or stops the service listener, the status indicator for the service is updated to show that it is started or stopped. This scenario helps you to throttle back traffic to a specific service as needed but leaves the containing application and other services in the application running. See service providers at the cell level using the administrative console. You can also reference service providers at the application level using the administrative console.

Note: You can only start or stop the service listener using the administrative console.

Another option is to use MBeans. With MBeans, you can invoke the startListener or stopListener operations in the EndpointCentralManager MBean or EndpointManager MBean to start or stop the listener service. The administrative console option does not expose the function of starting or stopping the listening state of a specific endpoint in a service. However, the MBeans option provides this capability. You can use scripting to invoke the MBean operations to start or stop the endpoint listener.

EndpointCentralManager MBean

There is an EndpointCentralManager MBean instance in the deployment manager, AdminAgent, and stand-alone server. The EndpointCentralManager MBean provides the administrative

convenience to start and stop the service or endpoint listeners across all the deployment targets such as the cluster members in a cluster. You do not have to know the target servers for the service application.

EndpointManager MBean

There is an EndpointManager MBean instance for each Web services application module in a server. This MBean instance is created when the application module is started. The MBean instance is deleted when the module is stopped. The MBean provides the start and stop operations to change the service and endpoint listener state. The MBean can also send a Java Management Extension (JMX) notification whenever the listener state has changed.

Viewing service providers at the cell level using the administrative console

You can use this administrative console page to view and manage your service providers at the cell level.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Services (JAX-WS).

About this task

You can view a list of your installed service providers, such as JAX-WS Web services providers in a cell. You can start or stop the service listener.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. In the navigation pane, expand **Services > Service providers**.
3. Locate the Web service of interest by name and the associated deployed asset in Deployed Asset column.
4. [Optional] Click the service of interest in the Name column to view the detail of the service and manage the policy sets and bindings for that service.
5. Click **Start Listener** or **Stop Listener** to start or stop the service listener. Starting the service listener makes all the endpoints for the service provider available and ready to receive messages. Stopping the listener makes all the endpoints for the service provider unavailable.

Results

When you finish this task, you have viewed the service providers at the cell level. You have also started or stopped the service listener.

What to do next

Proceed to view service providers at the application level using the administrative console.

Service providers collection at the cell level

Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

To view a service provider using this administrative console page, perform the following steps:

1. Expand **Services > Service Providers**.

2. Locate the service of interest by name in the Name column.
3. Click the service of interest in the Name column.
4. View the detail of the service.

To change the listening state of a service, perform the following steps:

1. Select the service of interest, and click **Start Listener** to start the listener service or click **Stop Listener** to stop the listener service. Starting the service listener makes all the endpoints for the service provider to be available and to receive messages. Stopping the listener makes all the endpoints for the service provider unavailable.
2. [Optional] Select the deployed asset link in the Deployed Asset column to access the deployed asset settings page.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

"Viewing service providers at the application level using the administrative console" on page 734
You can use this administrative console page to view and manage your service providers at the application level.

"Viewing service clients at the application level using the administrative console" on page 752
You can view your installed service clients for an application using this task.

"Viewing service providers at the cell level using the administrative console" on page 732
You can use this administrative console page to view and manage your service providers at the cell level.

"Viewing service clients at the cell level using the administrative console" on page 751
You can view all your service clients at the cell level using the administrative console.

Related reference

"Service client collection at the cell level" on page 751
Use this page to view and manage service clients at the cell level. Server-based Java API for XML-Based Web Services (JAX-WS) service clients are the only clients that are displayed in this view. Java API for XML-based RPC (JAX-RPC) service clients are not displayed in this view.

"Service providers collection at the application level" on page 735
Use this page to view and manage service providers at the application level.

"Service provider policy sets and bindings collection" on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Name:

Specifies the name of the service provider. The service, *QName* (`Java class javax.xml.namespace.QName`), is displayed when you hover your mouse pointer over the Name field.

Type:

Specifies the type of service, such as a JAX-WS Web service and Web Services Notification (WSN) client service. You can filter by the type of service. If your service was migrated from the Feature Pack for Web Services, you can only filter by the name of the service.

Deployed Asset:

Represents a Java Platform, Enterprise Edition (Java EE) application or a WS-Notification service point.

	Java EE application
	Web Services Notification client service point application

Status:

Indicates the status for the service listener.

	Service endpoints are listening.
	Service endpoints are running, but the listener control is not supported because of the characteristics of this service; for example, the application is installed on a Feature Pack for Web Services server.
	Either the deployed asset is partially started or the service endpoints are listening on some, but not all the target servers.
	The deployed asset is partially started and the listener control is not supported.
	Service endpoints are not listening, but the deployed asset is running.
	Service endpoints are not listening because the deployed asset is not running.
	Service endpoints are not listening because the deployed asset is not running. Listener control is not supported.
	Deployed asset is partially stopped. Service endpoints are not listening.
	Deployed asset is partially stopped. Service endpoints are not listening. Listener control is not supported because of the characteristics of this service; for example, the application is installed on a Feature Pack for Web Services server.
	Status cannot be determined.

Viewing service providers at the application level using the administrative console

You can use this administrative console page to view and manage your service providers at the application level.

Before you begin

Before completing this task, you need to install a Java API for XML-Based Web Service.

About this task

You can view a list of your service providers, such as JAX-WS Web services providers at the application level. You can start the service listener to make all the endpoints for the service to be available and to receive messages. You can also stop the service listener.

1. Open the administrative console.
2. To view the service providers, click **Applications > Application Types > WebSphere enterprise applications** >*Service_provider_application_instance* > **Service providers**.
3. [Optional] Click a module that contains a service to view the module information.
4. Depending on the current application status, click **Start Listener** or **Stop Listener** to start or stop the service listener. Starting the service listener makes all the endpoints for the service available and ready to receive messages. Stopping the listener makes all the endpoints for the service unavailable. See Endpoint and service listeners overview.

Results

When you complete this task, you have viewed and managed the service providers at the application level.

What to do next

You can proceed to managing policy sets using the administrative console. You can click on a service provider to manage the policy sets and bindings for that service.

Service providers collection at the application level

Use this page to view and manage service providers at the application level.

To view a service provider using this administrative console page, perform the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications** >*Service_provider_application_instance* > **Service providers**.
2. Locate the service of interest by name in the Name column.
3. Click the service of interest in the Name column.
4. View the detail of the service.

To change the listening state of a service, perform the following steps:

1. Select the service of interest, and click **Start Listener**, to start the service listener or **Stop Listener**, to stop the service listener. Starting the service listener makes all the endpoints for the service provider to be available and to receive messages. Stopping the listener makes all the endpoints for the service provider to be unavailable.
2. [Optional] Click a module name in the Module column to access the module detail page.

Related concepts

“Overview of service and endpoint listeners” on page 731
The administration function of the product is enhanced to support the service listener. This topics describes the service listener and endpoint listener.

Related tasks

“Viewing service providers at the application level using the administrative console” on page 734
You can use this administrative console page to view and manage your service providers at the application level.

“Viewing service providers at the cell level using the administrative console” on page 732
You can use this administrative console page to view and manage your service providers at the cell level.

Related reference

“Service providers collection at the cell level” on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

“Service provider policy sets and bindings collection” on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

Name:

Specifies the name of the service provider. The service *QName* (`Java class javax.xml.namespace.QName`) is displayed when you hover your mouse pointer over the Name field.

Type:

Specifies the type of service.

Module:

Specifies the name of the module that contains the service.

Status:

Indicates the status for the service listener.

	Service endpoints are listening.
	Service endpoints are running, but the listener control is not supported because of the characteristics of this service; for example, the application is installed on a Feature Pack for Web Services server.
	Either the deployed asset is partially started or the service endpoints are listening on some, but not all the target servers.
	The deployed asset is partially started and the listener control is not supported.
	Service endpoints are not listening, but the deployed asset is running.
	Service endpoints are not listening because the deployed asset is not running.
	Service endpoints are not listening because the deployed asset is not running. Listener control is not supported.
	Deployed asset is partially stopped. Service endpoints are not listening.
	Deployed asset is partially stopped. Service endpoints are not listening. Listener control is not supported because of the characteristics of this service; for example, the application is installed on a Feature Pack for Web Services server.
	Status cannot be determined.

Viewing the detail of a service provider and managing policy sets using the administrative console

Use this administrative console task to view the detail of your service provider and to manage the policy sets for the service, its endpoints and operations.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Services (JAX-WS) artifacts.

About this task

You have developed a Web service that contains all the necessary artifacts and deployed your Web services application into your application server instance. Now, you can attach or detach policy sets and manage the associated bindings.

The policy set information is displayed in the Attached Policy Set column. If a policy set is directly attached, then the policy set name appears; for example, `WS-I RSP` is displayed. If there is no policy set attached, and a policy set is attached at a higher level, then the word *inherited* in parentheses is appended to the policy set name, as the following example demonstrates: `WS-I RSP (inherited)`. If there is no policy set attached directly or at a higher level, then `None` is displayed.

Every attachment of a policy set to a service artifact has an assigned binding. The binding information is displayed in the Binding column. The Binding column can contain the following values:

- Not applicable. There is no policy set attached, either directly or to a higher level service resource.
- *Binding_name* or Default. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. Default is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* (inherited) or Default (inherited). A service resource inherits the bindings from an attachment to a higher level resource.

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have very limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings. General provider policy set bindings might also be used for trust service attachments.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. In the navigation pane, expand **Services > Service providers >***Service_provider_application_instance* **Service providers**.
3. Select one or more service, endpoints and operations of interest and view the associated service, endpoints and operations.

4. Do one or more of the following actions:
   - Click **Attach**, to attach a policy to a selected service, endpoint or operation.
   - Click **Detach**, to detach a policy set from a list of attached policy sets for a service, endpoint or operation.
5. Click **Assign Binding** to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options:
   - Default
   - New Application Specific Binding
   - Provider sample

   Default

   Specifies the default binding for the selected service, endpoint or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the Default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used.

   For any policy set attachment, the runtime checks to see if the attachment includes a binding. If so, it uses that binding. If not, the runtime checks in the following order and uses the first available default binding:

   a. Default general bindings for the server
   b. Default general bindings for the domain that the server resides
   c. Default general bindings for the global security domain

   New Application Specific Binding

   Select this option to create a new application specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached.

   Provider sample

   Select this option to use the provider sample binding.
6. To close the drop down list for the assign binding action, click **Assign Binding**.

Results

When you finish this task, a policy set is attached, detached or a binding is assigned to the service artifact.

Example

You have configured a service provider, `EchoService12` in the application instance, WSSampleServicesSei. Now you want to attach the WS-Security policy to the `EchoService12Port` endpoint of the `EchoService12` service provider. First locate **`EchoService12`** in the Services > Service providers collection. Click the **`EchoService12`** service provider. Select the check box for the `columoService12Port` resource. Click **Attach** and select **WSSecurity** default policy from the list. Click **Save**, to save your changes to the master configuration.

What to do next

You can now proceed to manage policy sets and bindings for service providers at the application level using the administrative console.

Service provider settings

Use the Service provider settings page to manage the settings for your service providers. You can attach and detach policy sets to an application, its service, endpoints or operations. You can create new bindings,

or use bindings that you have already created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

Use the **WSDL document** link to view the Web Services Description Language (WSDL) for the service. The Application and Module links provide access to the application and module settings page.

To view this administrative console page, click **Services** → **Service providers** → ***service_provider_instance***.

You can also view this page by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_provider_application_instance*** → **Service providers** → ***service_provider_instance***.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

“Managing policy sets and bindings for service providers at the application level using the administrative console” on page 743
Use this administrative console task to manage policy sets for an application or its services, endpoints, and operations.

“Viewing detail of a service client and managing policy sets using the administrative console” on page 754
Use this administrative console task to view the detail of your service client reference and to manage the policy sets for the service, its endpoints and operations.

“Viewing service clients at the application level using the administrative console” on page 752
You can view your installed service clients for an application using this task.

Related reference

“Service provider policy sets and bindings collection” on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

“Service providers collection at the cell level” on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

“Service providers collection at the application level” on page 735
Use this page to view and manage service providers at the application level.

“Service client settings” on page 756
Use this administrative console page to manage the settings for your service clients. You can attach and detach policy sets to a service, its endpoints, or operations. You can select default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

“Service clients collection at the application level” on page 753
Use this page to view and manage service clients at the application level.

“Policy sharing settings” on page 723
Use this pane to view and change whether the policy configuration of a Web services service provider is shared. You can configure the service provider to include the policy configuration in its Web Service Description Language (WSDL) so that it can be accessed using an HTTP Get request, or published. You can also make the policy configuration available to a Web Services Metadata Exchange (WS-MetadataExchange) request.

Enterprise application settings
Use this page to configure an enterprise application.

“Web module deployment settings” on page 34
Use this page to configure an instance of Web module deployment.

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Service provider:

Specifies the name of the service provider that is displayed.

Policy Set Attachments:

Service/Endpoint/Operation:

Specifies the name of the service provider, endpoint, or operation that is contained in a service.

Attached Policy Set:

Specifies the policy set that is attached to a service provider, endpoint, or operation.

The Attached Policy Set column can contain the following values:

- **None**. No policy set is attached, either directly or to a higher-level service resource.
- ***Policy_set_name***. The name of the policy set that is attached directly to the service resource, for example, `WS-I RSP`.
- ***Policy_set_name*** **(inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding:

Specifies the binding configuration that is available for a service provider, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- ***Binding_name*** or **Default**. The binding name is displayed if a policy set is attached directly and an application specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- ***Binding_name*** **(inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for

service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

Policy Sharing:

Specifies whether the service provider can share its current policy configuration.

The Policy sharing column can contain the following values:

- **Not applicable**. The resource does not have a policy set attached, so there is no policy configuration to share.
- **Disabled**. The policy set of the resource cannot be shared. This is the default setting if a policy set is attached to a service.
- **Enabled**. The policy set of the resource can be shared.

When the value in the column is a link, click the link to view or change settings about how the policy configuration can be shared.

For a service, if the policy set is inherited from the parent application, the policy sharing value is also inherited, and you cannot change it. The value is not a link and it is followed by the term `inherited` in parentheses.

For an endpoint or operation, the value is not a link and it is followed by the term `inherited` in parentheses. The setting is inherited from the parent application or service and you cannot change it.

Buttons:

Attach Policy Set	Click this button to view a list of policy sets available for attachment to the selected service, endpoint, or operation. Select a policy set from the list to attach and it is attached to the selected service, endpoint, or operation. To close the menu list, click **Attach Policy Set**.
Detach Policy Set	Click this button to detach a policy set from a selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper level service resource, the **Attached Policy Set** column displays **None** and the **Binding** column displays **Not Applicable**. If there is a policy set attached to an upper level service resource, the **Attached Policy Set** column displays***policy_set_name*** **(inherited)** and the binding used for the upper level attachment is applied. The binding name is displayed followed by **(inherited)**.

Assign Binding	Click this button to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options: • Default • New Application Specific Binding • Provider sample **Default** Specifies the default binding for the selected service, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Provider sample** Select this option to use the Provider sample binding. To close the menu list, click **Assign Binding**.

Managing policy sets and bindings for service providers at the application level using the administrative console

Use this administrative console task to manage policy sets for an application or its services, endpoints, and operations.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Services (JAX-WS) and attach a policy set to each Web service.

About this task

You have developed a Web service that contains all the necessary artifacts and deployed your Web services application into your application server instance. Now, you can attach or detach policy sets and manage the associated bindings.

The policy set information is displayed in the Attached Policy Set column. If a policy set is directly attached, then the policy set name appears; for example, `WS-I RSP` is displayed. If there is no policy set attached, and a policy set is attached at a higher level, then the word *inherited* in parentheses is appended to the policy set name, as the following example demonstrates: `WS-I RSP (inherited)`. If there is no policy set attached directly or at a higher level, then `None` is displayed.

Every attachment of a policy set to a service artifact has an assigned binding. The binding information is displayed in the Binding column. The Binding column can contain the following values:

- Not applicable. There is no policy set attached, either directly or to a higher level service resource.
- *Binding_name* or Default. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. Default is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* (inherited) or Default (inherited). A service resource inherits the bindings from an attachment to a higher level resource.

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have very limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings. General provider policy set bindings might also be used for trust service attachments.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. **Applications > Application Types > WebSphere enterprise applications** >*Service_provider_application_instance* > **Service provider policy sets and bindings**.
3. Select the check box next to a service resource of interest.
4. Click **Attach** to attach a policy set to an application, service, endpoint or operation.

5. [Optional] Click **Detach** to detach a policy set from a list of attached policy sets for an application, service provider, endpoint or operation.
6. Click **Assign Binding** to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options:
   - Default
   - New Application Specific Binding
   - Provider sample

   Default

   Specifies the default binding for the selected service, endpoint or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the Default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used.

   For any policy set attachment, the runtime checks to see if the attachment includes a binding. If so, it uses that binding. If not, the runtime checks in the following order and uses the first available default binding:

   a. Default general bindings for the server
   b. Default general bindings for the domain that the server resides
   c. Default general bindings for the global security domain

   New Application Specific Binding

   Select this option to create a new application specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached.

   Provider sample

   Select this option to use the provider sample binding.
7. To close the drop down list for the assign binding action, click **Assign Binding**.

Results

When you finish this task, a policy set is attached or detached, and a binding is assigned to the service artifact.

Example

If you have configured a service provider application instance, `app1` and you want, for example, to attach the`Username WSSecurity default` policy set to your application, first locate `app1` application in the **Applications > Application Types > WebSphere enterprise applications**. Click `app1` > **Service provider policy sets and bindings** under the Web Services Properties section. Select the check box next to the `app1` service application. Click **Attach** and select `Username WSSecurity default` policy set. Click **Save**, to save your changes to the master configuration. You are returned to the **Applications > Application Types > WebSphere enterprise applications** page.

To assign a binding to the attached policy set, click `app1` > **Service provider policy sets and bindings**. Select the check box next to the `app1` service application and click **Assign Binding**. Select **Provider sample** binding from the list. Click **Save**, to save your changes to the master configuration.

What to do next

You can proceed to view service providers at the cell level using the administrative console.

Service provider policy sets and bindings collection

Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

This page provides detail information for an application and its associated Web service providers, endpoints, and operations. You can view and manage policy set attachments and bindings information using this page.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***Service_provider_application_instance*** → **Service provider policy sets and bindings**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

"Managing policy sets and bindings for service providers at the application level using the administrative console" on page 743
Use this administrative console task to manage policy sets for an application or its services, endpoints, and operations.

"Defining and managing policy set bindings" on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

"Defining and managing service client or provider bindings" on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

"Viewing detail of a service client and managing policy sets using the administrative console" on page 754
Use this administrative console task to view the detail of your service client reference and to manage the policy sets for the service, its endpoints and operations.

"Viewing service clients at the application level using the administrative console" on page 752
You can view your installed service clients for an application using this task.

Related reference

"Policy sharing settings" on page 723
Use this pane to view and change whether the policy configuration of a Web services service provider is shared. You can configure the service provider to include the policy configuration in its Web Service Description Language (WSDL) so that it can be accessed using an HTTP Get request, or published. You can also make the policy configuration available to a Web Services Metadata Exchange (WS-MetadataExchange) request.

"Service providers collection at the cell level" on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

"Service providers collection at the application level" on page 735
Use this page to view and manage service providers at the application level.

"Service client settings" on page 756
Use this administrative console page to manage the settings for your service clients. You can attach and detach policy sets to a service, its endpoints, or operations. You can select default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

"Service client policy set and bindings collection" on page 763
Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console

panel. The preference settings vary from one administrative console panel to another.

Application/Service/Endpoint/Operation:

Specifies the name of the application and the associated service providers, endpoints or operations.

The Application/Service/Endpoint/Operation column lists the service application and the service providers, endpoints, or operations that the application contains.

Attached Policy Set:

Specifies the policy set that is attached to a service provider, endpoint, or operation.

The Attached Policy Set column can contain the following values:

- **None**. No policy set is attached, either directly or to a higher-level service resource.
- ***Policy_set_name***. The name of the policy set that is attached directly to the service resource, for example, WS-I RSP.
- ***Policy_set_name* (inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher-level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Binding:

Specifies the binding configuration that is available for a service provider, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. No policy set is attached, either directly or to a higher-level service resource.
- ***Binding_name*** or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- ***Binding_name* (inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

Policy Sharing:

Specifies whether the service provider can share its current policy configuration.

The Policy sharing column can contain the following values:

- **Not applicable**. The resource does not have a policy set attached, so there is no policy configuration to share.
- **Disabled**. The policy set of the resource cannot be shared. This is the default setting if a policy set is attached to a service.
- **Enabled**. The policy set of the resource can be shared.

When the value in the column is a link, click the link to view or change settings about how the policy configuration can be shared.

For a service, if the policy set is inherited from the parent application, the policy sharing value is also inherited, and you cannot change it. The value is not a link and it is followed by the term `inherited` in parentheses.

For an endpoint or operation, the value is not a link and it is followed by the term `inherited` in parentheses. The setting is inherited from the parent application or service and you cannot change it.

Buttons:

Attach Policy Set	Click this button to view a list of policy sets available for attachment to the selected service, endpoint, or operation. Select a policy set from the list to attach and it is attached to the selected service, endpoint, or operation. To close the menu list, click **Attach Policy Set**.
Detach Policy Set	Click this button to detach a policy set from a selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper level service resource, the **Attached Policy Set** column displays **None** and the **Binding** column displays **Not Applicable**. If there is a policy set attached to an upper level service resource, the **Attached Policy Set** column displays ***policy_set_name*** **(inherited)** and the binding used for the upper level attachment is applied. The binding name is displayed followed by **(inherited)**.

Assign Binding	Click this button to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options: • Default • New Application Specific Binding • Provider sample **Default** Specifies the default binding for the selected service, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Provider sample** Select this option to use the Provider sample binding. To close the menu list, click **Assign Binding**.

Viewing WSDL document using the administrative console

You can locate and view a Web Services Description Language (WSDL) document from the administrative console.

Before you begin

Before completing this task, you need to install or deploy a Java API for XML Web Services (JAX-WS). Read about the JAX-WS application deployment model.

About this task

JAX-WS integration with the WebSphere Application Server provides the option to view a Web Services Description Language (WSDL) document that is associated with your Web service using the administrative console.

A Web service application that contains all the necessary artifacts has been developed and deployed to the Application Server. Now you can view the WSDL document for each service using the administrative console.

1. Open the administrative console.
2. In the navigation pane, expand **Services > Service providers**.
3. Select a Web service of interest in the **Service providers** collection view.
4. Click the **WSDL document** link to view the WSDL document.

Results

When you finish this task, you have viewed the WSDL document for a service artifact.

What to do next

You can attach and detach policy sets and configure bindings information for your application. See managing policy sets and bindings for service providers at the application level using the administrative console.

Viewing service clients at the cell level using the administrative console

You can view all your service clients at the cell level using the administrative console.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Services (JAX-WS).

About this task

You can view a list of your service clients, such as JAX-WS Web services clients using the administrative consolel.The Name column displays the service client references. The Deployed Asset column displays the name of the deployed asset that contains the service client.

1. Open the administrative console.
2. In the navigation pane, expand **Services > Service clients**.
3. View the Web service client of interest in the Name column and the associated application in the Deployed Asset column. For JAX-WS (WSN) references, the name of the containing WS-Notification service that is a part of the System Integration (SI) is displayed. Hovering over this name displays the Bus:<busName>.

Results

When you complete this task, you have viewed the service clients at the cell level.

What to do next

Proceed to view service clients at the application level using the administrative console. Click each service to see and manage the policy sets and bindings associated with a particular service.

Service client collection at the cell level

Use this page to view and manage service clients at the cell level. Server-based Java API for XML-Based Web Services (JAX-WS) service clients are the only clients that are displayed in this view. Java API for XML-based RPC (JAX-RPC) service clients are not displayed in this view.

To view this administrative console page, click **Services > Service clients**. Select the service name link in the Name column to access the service client settings page. Select the deployed asset link in the Deployed Asset column to access the deployed asset settings page.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

“Viewing service clients at the cell level using the administrative console” on page 751
You can view all your service clients at the cell level using the administrative console.

“Viewing service providers at the cell level using the administrative console” on page 732
You can use this administrative console page to view and manage your service providers at the cell level.

“Viewing service providers at the application level using the administrative console” on page 734
You can use this administrative console page to view and manage your service providers at the application level.

“Viewing service clients at the application level using the administrative console”
You can view your installed service clients for an application using this task.

Related reference

“Service client settings” on page 756
Use this administrative console page to manage the settings for your service clients. You can attach and detach policy sets to a service, its endpoints, or operations. You can select default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Name:

Specifies the name of the service client, which is the name of the service provider that is referenced by the client. The full `QName`, Java class `javax.xml.namespace.QName`, is displayed when you hover the mouse pointer over the **Name** field.

Type:

Specifies the type of service such as a JAX-WS client service and Web Services Notification (WSN) client service. You can filter by the type of service.

Deployed Asset:

Represents a Java Platform, Enterprise Edition (Java EE) application or a WS-Notification service. For JAX-WS or WSN clients, the name of the containing WS-Notification service that is a part of the System Integration (SI) is displayed. Hovering over this name displays the Bus:<busName>.

	Java EE application
	WS-Notification service

Viewing service clients at the application level using the administrative console

You can view your installed service clients for an application using this task.

Before you begin

Before completing this task, you need to install a Java API for XML-based Web Service.

About this task

You can view a list of your service client references.

Your application server instance can have one or more applications deployed on it that contain service clients. Use this task enables you to view the service names that are referenced in an application.

1. Open the administrative console.
2. View the service clients in an application by expanding **Applications > Application Types > WebSphere enterprise applications** >*Service_client_application_instance* > **Service clients**.
3. Select a client from the Name column to view the client and to manage the policy sets and bindings associated with it.
4. [Optional] Select a module name from the Module column to access the module detail page.

Results

When you finish this task, you have viewed service clients at the application level.

What to do next

You can now proceed to managing policy sets using the administrative console.

Service clients collection at the application level

Use this page to view and manage service clients at the application level.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications** >*Service_client_application_instance* > **Service client**.

Related tasks

"Viewing service clients at the application level using the administrative console" on page 752
You can view your installed service clients for an application using this task.

"Viewing service providers at the application level using the administrative console" on page 734
You can use this administrative console page to view and manage your service providers at the application level.

"Viewing service providers at the cell level using the administrative console" on page 732
You can use this administrative console page to view and manage your service providers at the cell level.

"Viewing service clients at the cell level using the administrative console" on page 751
You can view all your service clients at the cell level using the administrative console.

Related reference

"Service providers collection at the application level" on page 735
Use this page to view and manage service providers at the application level.

"Service client collection at the cell level" on page 751
Use this page to view and manage service clients at the cell level. Server-based Java API for XML-Based Web Services (JAX-WS) service clients are the only clients that are displayed in this view. Java API for XML-based RPC (JAX-RPC) service clients are not displayed in this view.

"Service providers collection at the cell level" on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

"Service client settings" on page 756
Use this administrative console page to manage the settings for your service clients. You can attach and detach policy sets to a service, its endpoints, or operations. You can select default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

Name:

Specifies the name of the service client. The service *QName*, `Java class javax.xml.namespace.QName`, is displayed when you hover your mouse pointer over the **Name** field.

Type:

Specifies the type of service such as a Java™ API for XML-Based Web Services (JAX-WS) Web service. You can filter by the type of service.

Module:

Specifies the name of the module that contains the service. Selecting a module name accesses the module detail page.

Viewing detail of a service client and managing policy sets using the administrative console

Use this administrative console task to view the detail of your service client reference and to manage the policy sets for the service, its endpoints and operations.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Service (JAX-WS) Web service and attach a policy set to that Web service.

About this task

You have developed a Web service that contains all the necessary artifacts and deployed your Web services application into your application server instance. Now, you can attach or detach policy sets and manage the associated bindings.

The policy set information is displayed in the Attached Policy Set column. If a policy set is directly attached, then the policy set name appears; for example, `WS-I RSP` is displayed. If there is no policy set attached, and a policy set is attached at a higher level, then the word *inherited* in parentheses is appended to the policy set name, as the following example demonstrates: `WS-I RSP (inherited)`. If there is no policy set attached directly or at a higher level, then `None` is displayed.

Every attachment of a policy set to a service artifact has an assigned binding. The binding information is displayed in the Binding column. The Binding column can contain the following values:

- Not applicable. There is no policy set attached, either directly or to a higher level service resource.
- *Binding_name* or Default. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. Default is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* (inherited) or Default (inherited). A service resource inherits the bindings from an attachment to a higher level resource.

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have very limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have

added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service clients policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general client policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. In the navigation pane, click **Applications > Application Types > WebSphere enterprise applications** >*Service_client_application_instance* > **Service clients**.
3.
4. Select one or more service, endpoints and operations of interest and view the associated service, endpoints and operations.
5. You can perform any of the following actions:
   - Click **Attach**, to attach a policy set to a selected service, endpoint or operation.
   - Click, **Detach**, to detach a policy set from a list of attached policy sets for a service, endpoint or operation. The service name is the service client reference in the application.
6. Click **Assign Binding** to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options:
   - Default
   - New Application Specific Binding
   - Client sample

   Default

   Specifies the default binding for the selected service, endpoint or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the Default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used.

   For any policy set attachment, the runtime checks to see if the attachment includes a binding. If so, it uses that binding. If not, the runtime checks in the following order and uses the first available default binding:

a. Default general bindings for the server
b. Default general bindings for the domain that the server resides
c. Default general bindings for the global security domain

New Application Specific Binding

Select this option to create a new application specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached.

Client sample

Select this option to use the client sample binding.

7. To close the drop down list for the assign binding action, click **Assign Binding**.

Results

When you finish this task, a policy set is attached, detached or a binding is assigned to the service artifact.

Example

You have configured a service client reference, `EchoService12` in the application instance, WSSampleClientSei. Now you want to attach the WSSecurity default policy to the `EchoService12Port` endpoint of the `EchoService12` service client reference. First locate `EchoService12` in the Services > Service clients collection. Click the **`EchoService12`** service client reference. Select the check box for the `EchoService12Port` resource and click **Attach**. Select the **WSSecurity** default policy from the list. Click **Save**, to save your changes to the master configuration.

What to do next

You can now proceed to manage policy sets and bindings for service clients at the application level using the administrative console.

Service client settings

Use this administrative console page to manage the settings for your service clients. You can attach and detach policy sets to a service, its endpoints, or operations. You can select default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

This service client page displays configuration information for a service client and the associated endpoints and operations. You can view and manage policy set attachments, bindings information, and whether the client uses the policy of the service provider.

The Application and Module links provide access to the application and module settings page.

To view this administrative console page, click **Services** → **Service clients** → ***service_client_instance***.

You can also view this page by clicking **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_client_application_instance*** → **Service clients** → ***service_client_instance***.

Related tasks

“Viewing detail of a service client and managing policy sets using the administrative console” on page 754
Use this administrative console task to view the detail of your service client reference and to manage the policy sets for the service, its endpoints and operations.

Related reference

“Service provider policy sets and bindings collection” on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

“Service client policy set and bindings collection” on page 763
Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

“Service providers collection at the cell level” on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

“Service providers collection at the application level” on page 735
Use this page to view and manage service providers at the application level.

“Service client collection at the cell level” on page 751
Use this page to view and manage service clients at the cell level. Server-based Java API for XML-Based Web Services (JAX-WS) service clients are the only clients that are displayed in this view. Java API for XML-based RPC (JAX-RPC) service clients are not displayed in this view.

“Policies applied settings” on page 727
Use this pane to view and change whether the policy configuration of a WebSphere Application Server service client is configured dynamically, based on the policies supported by its service provider. You can view or change how the client obtains the policy of the service provider; the client can use an HTTP Get request or a Web Services Metadata Exchange (WS-MetadataExchange) request. You can specify a policy set and binding to provide message-level security for WS-MetadataExchange.

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Service client:

Specifies the name of the service client that is displayed.

Policy Set Attachments:

Service/Endpoint/Operation:

Specifies the name of the service client, endpoints or operations. The full `QName` (Java class `javax.xml.namespace.QName`) is displayed when you hover the mouse pointer over a service client name.

Attached Client Policy Set:

Specifies the policy set that is attached to the service client, endpoints or operations.

The Attached Client Policy Setcolumn can contain the following values:

- **None**. No policy set is attached, either directly or to a higher level service resource.
- ***policy_set_name***. The name of the policy set that is attached directly to the service resource, for example, `WS-I RSP`.
- ***policy_set_name* (inherited)**. The name of the policy set that is not attached directly to a service resource, but that is attached to a higher level service resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Policies Applied:

Specifies the policies that are applied to the resource.

The Policies Applied column can contain the following values:

- **None**. No policies are applied to the service. This is the default setting if there is no policy set attached to the client.
- **Client only**. The client policy set is applied to the service. This is the default setting if a policy set is attached to the client.
- **Provider only**. The policy configuration of the service provider is applied to the service, as long as the client can support those policies.
- **Client and provider**. A policy that is based on both the client policy set and the policy of the service provider is applied to the service.

When the value in the column is a link, click the link to view or change settings about how the policies are applied.

For a service, if the value in the column is a link followed by the word `inherited` in parentheses, this shows a setting that is inherited from the parent application. You can click the link to change the setting for the service.

For an endpoint or operation, the value is not a link and it is followed by the word `inherited` in parentheses. The setting is inherited from the parent application or service and you cannot change it. If there is no applied policy, the entries in the column are None or None (inherited).

Binding:

Specifies the binding information available for a service client, endpoint, or operation.

The Binding column can contain the following values:

- **Not applicable**. There is no policy set attached, either directly or to a higher level service resource.
- ***Binding_name*** or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. Default is displayed if a policy set is attached directly but the service resource uses the default bindings.
- ***Binding_name* (inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

Buttons:

Attach Client Policy Set	Click this button to view a list of policy sets available for attachment to the selected service, endpoint, or operation. Select a policy set from the list to attach and it is attached to the selected service, endpoint, or operation. To close the menu list, click **Attach Client Policy Set**.

Detach Client Policy Set	Click this button to detach a policy set from a selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper level service resource, the Attached Client Policy Set column displays **None** and the Binding column displays **Not Applicable**. If there is a policy set attached to an upper level service resource, the Attached Client Policy Set column displays ***policy_set_name*** **(inherited)** and the binding used for the upper level attachment is applied. The binding name is displayed followed by **(inherited)**.
Assign Binding	**Default** Specifies the default binding for the selected service, endpoint, or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain in which the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Provider sample** Select this option to use the Provider sample binding. To close the menu list, click **Assign Binding**.

Managing policy sets and bindings for service clients at the application level using the administrative console

Use this administrative console task to manage policy sets for service clients applications or its services, endpoints, or operations.

Before you begin

Before completing this task, you need to install one or more Java API for XML-Based Web Services (JAX-WS) applications.

About this task

You have developed a Web service that contains all the necessary artifacts and deployed your Web services application into your application server instance. Now, you can attach or detach policy sets and manage the associated bindings.

The policy set information is displayed in the Attached Policy Set column. If a policy set is directly attached, then the policy set name appears; for example, `WS-I RSP` is displayed. If there is no policy set attached, and a policy set is attached at a higher level, then the word *inherited* in parentheses is appended to the policy set name, as the following example demonstrates: `WS-I RSP (inherited)`. If there is no policy set attached directly or at a higher level, then `None` is displayed.

Every attachment of a policy set to a service artifact has an assigned binding. The binding information is displayed in the Binding column. The Binding column can contain the following values:

- Not applicable. There is no policy set attached, either directly or to a higher level service resource.
- *Binding_name* or Default. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. Default is displayed if a policy set is attached directly but the service resource uses the default bindings.
- *Binding_name* (inherited) or Default (inherited). A service resource inherits the bindings from an attachment to a higher level resource.

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have very limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service clients policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general client policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. In the navigation pane, click **Applications > Application Types > WebSphere enterprise applications** >*Service_client_application_instance* > **Service client policy sets and bindings**.
3. Select the check box next to the **Application/Service/Endpoint/Operation** column. The service name is the service client reference in the application.
4. Click **Attach** to attach a policy set to an application, service, endpoint or operation.
5. [Optional] Click **Detach** to detach a policy set from a list of attached policy sets for an application, service client, endpoint or operation.
6. Click **Assign Binding** to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options:
   - Default
   - New Application Specific Binding
   - Client sample

   Default

   Specifies the default binding for the selected service, endpoint or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the Default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used.

   For any policy set attachment, the runtime checks to see if the attachment includes a binding. If so, it uses that binding. If not, the runtime checks in the following order and uses the first available default binding:

   a. Default general bindings for the server

   b. Default general bindings for the domain that the server resides

   c. Default general bindings for the global security domain

   New Application Specific Binding

   Select this option to create a new application specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached.

   Client sample

   Select this option to use the client sample binding.

Results

When you finish this task, a policy set is attached or detached, and a binding is assigned to the service artifact.

Example

If you have configured a service client application instance, `app1` and you want, for example, to attach the Username WSSecurity default policy set to your application, first locate `app1` application in the **Applications > Application Types > WebSphere enterprise applications**. Click `app1` > **Service client policy sets and bindings** under the Web Services Properties section. Select the check box next to the `app1` service application. Click **Attach** and select **Username WSSecurity default** policy set. Click **Save**, to save your changes to the master configuration.

To assign a binding to the attached policy set, click `app1` > **Service client policy sets and bindings**. Select the check box next to the `app1` service application and click **Assign Binding**. Select **client sample** binding from the list. Click **Save**, to save your changes to the master configuration.

What to do next

You can proceed to view service clients at the cell level using the administrative console.

Service client policy set and bindings collection

Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

This page displays detail information for an application and its associated Web service clients, endpoints, and operations. You can view and manage policy set attachments and bindings information using this page.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***service_client_application_instance*** → **Service client policy sets and bindings**.

This console page can also be viewed for WS-Notification service clients by clicking one of the following paths:

- **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**
- **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional properties] Outbound request policy sets and bindings**

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

“Managing policy sets and bindings for service providers at the application level using the administrative console” on page 743
Use this administrative console task to manage policy sets for an application or its services, endpoints, and operations.

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Defining and managing service client or provider bindings” on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

Related reference

Web services: Client security bindings collection
Use this page to view a list of application-level, client-side binding configurations for Web services security. These bindings are used when a Web service is a client to another Web service.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

“Search attached applications collection” on page 957
Use this page to search for applications and other resources that are attached to a specific policy set or to search for applications and other resources that have attached service resources.

“Policies applied settings” on page 727
Use this pane to view and change whether the policy configuration of a WebSphere Application Server service client is configured dynamically, based on the policies supported by its service provider. You can view or change how the client obtains the policy of the service provider; the client can use an HTTP Get request or a Web Services Metadata Exchange (WS-MetadataExchange) request. You can specify a policy set and binding to provide message-level security for WS-MetadataExchange.

“Service providers collection at the cell level” on page 732
Use this page to view and manage service providers at the cell level. Java Application Programming Interface (API) for XML-Based Web Services (JAX-WS) service providers are displayed in this view. Java API for XML Remote Procedure Call (JAX-RPC) service providers are not displayed in this view.

“Service providers collection at the application level” on page 735
Use this page to view and manage service providers at the application level.

“Service provider policy sets and bindings collection” on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Application/Service/Endpoint/Operation:

Specifies the name of the application and the associated service client, endpoints, or operations. For WS-Notification service clients, the first entry is associated with the WS-Notification service, not an application.

Attached Client Policy Set:

Specifies the policy set that is attached to the application, service clients, endpoints, or operations.

The Attached Client Policy Set column can contain the following values:

- None. No policy set is attached directly, or is attached at an upper level.
- ***policy_set_name***. The name of the policy set that is directly attached, for example, WS-I RSP.
- *policy_set_name* (inherited). A policy set is not directly attached to the resource, but a policy set is attached to a higher-level resource.

When the value in the column is a link, click the link to view or change settings about the attached policy set.

Policies Applied:

Specifies the policies that are applied to the resource. This column is not applicable and is not shown for WS-Notification service clients.

The Policies Applied column can contain the following values:

- **None**. No policies are applied to the application or service. This is the default setting if there is no policy set attached to the client.
- **Client only**. The client policy set is applied to the application or service. This is the default setting if a policy set is attached to the client.
- **Provider only**. The policy configuration of the service provider is applied to the application or service, as long as the client can support those policies.
- **Client and provider**. A policy that is based on both the client policy set and the policy of the service provider is applied to the application or service.

When the value in the column is a link, click the link to view or change settings about how the policies are applied.

For a service, if the value in the column is a link followed by the word `inherited` in parentheses, this shows a setting that is inherited from the parent application. You can click the link to change the setting for the service.

For an endpoint or operation, the value is not a link and it is followed by the word `inherited` in parentheses. The setting is inherited from the parent application or service and you cannot change it.

Binding:

Specifies the name of the binding associated with a policy set.

The Binding column can contain the following values:

- **Not applicable**. There is no policy set attached, either directly or to a higher-level service resource.
- ***Binding_name*** or **Default**. The binding name is displayed if a policy set is attached directly and an application-specific binding or a general binding is assigned, for example, `MyBindings1`. **Default** is displayed if a policy set is attached directly but the service resource uses the default bindings.
- ***Binding_name*** **(inherited)** or **Default (inherited)**. A service resource inherits the bindings from an attachment to a higher-level resource.

When the value in the Binding column is a link, click the link to view or change settings about the binding.

About policy set bindings

In this release, there are two types of bindings: application-specific bindings and general bindings.

Application-specific bindings

You can create application-specific bindings only at a policy set attachment point. These bindings are specific to, and constrained by, the characteristics of the defined policy. Application-specific bindings can provide configuration for advanced policy requirements such as multiple signatures; however, these bindings are reusable only within an application. Also, application-specific bindings have very limited reuse across policy sets.

When you create an application-specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding, and fully configure the bindings for each policy that you add. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application-specific bindings.

For service providers, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service providers policy sets and bindings collection page, for service provider resources that have an attached policy set. Similarly, for service clients, you can create application-specific bindings only by selecting **Assign Binding > New Application Specific Binding**, on the Service clients policy sets and bindings collection page, for service client resources that have an attached policy set.

General bindings

You can configure general bindings to be used across a range of policy sets and they can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they cannot provide configuration for advanced policy requirements such as multiple signatures. There are two types of general bindings: general provider policy set bindings and general client policy set bindings.

You can create general provider policy set bindings by clicking **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel, or by clicking **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. For details about defining and managing service client or provider bindings, see the related links. General provider policy set bindings might also be used for trust service attachments.

Buttons:

Attach Client Policy Set	Click this button to view a list of policy sets available for attachment to the selected service, endpoint, or operation. Select a policy set from the list to attach and it is attached to the selected service, endpoint, or operation. To close the menu list, click **Attach Client Policy Set**.

Detach Client Policy Set	Click this button to detach a policy set from a selected service, endpoint, or operation. After the policy set is detached, if there is no policy set attached to an upper level service resource, the Attached Client Policy Set column displays **None** and the Binding column displays **Not Applicable**. If there is a policy set attached to an upper level service resource, the Attached Client Policy Set column displays ***policy_set_name*** **(inherited)** and the binding used for the upper level attachment is applied. The binding name is displayed followed by **(inherited)**.
Assign Binding	Click this button to select from a list of available bindings for the selected policy set attachment. All the bindings are listed along with the following options: • Default • New Application Specific Binding • Client sample **Default** Specifies the default binding for the selected service, endpoint or operation. You can specify client and provider default bindings to be used at the cell level or global security domain level, for a particular server, or for a security domain. The default bindings are used when an application-specific binding has not been assigned to the attachment. When you attach a policy set to a service resource, the binding is initially set to the default. If you do not specifically assign a binding to the attachment point using this Assign Binding action, the default specified at the nearest scope is used. For any policy set attachment, the run time checks to see if the attachment includes a binding. If so, it uses that binding. If not, the run time checks in the following order and uses the first available default binding: 1. Default general bindings for the server 2. Default general bindings for the domain that the server resides 3. Default general bindings for the global security domain **New Application Specific Binding** Select this option to create a new application-specific binding for the policy set attachments. The new binding you create is used for the selected resources. If you select more than one resource, ensure that all selected resources have the same policy set attached. **Client sample** Select this option to use the Client sample binding. To close the menu list, click **Assign Binding**.

Viewing Web services deployment descriptors in the administrative console

You can view the Web services client and server deployment descriptors for a deployed Web services application. You can view the bindings in the deployment descriptors.

Before you begin

Before you can view the deployment descriptors, you need to complete all tasks required to implement a JAX-RPC Web service, create the deployment descriptor templates that were generated by the WSDL2Java command-line tool, configure the deployment descriptors and bindings, and deploy the Web service application into WebSphere Application Server. Review the implementing Web services application documentation for more information on developing and implementing Web services.

About this task

Deployment descriptors contain the information that is needed by a Web services client to communicate with the server for which the Web services is installed. This information is added to the deployment descriptor templates after a Web service is developed or an existing Web service is located. The data that you can view in the deployment descriptor includes the following:

- The Web service description including the name, WSDL file, WSDL file location and the mapping file.
- The port description, including the port component name, the WSDL port, the service endpoint interface that indicate the service's bindings, and the EJB that is used to implement the Web service.

After you have developed a Web service that contains all the necessary artifacts, created the deployment descriptors from the deployment descriptor templates, configured the deployment descriptors, and deployed the Web services application into WebSphere Application Server; now you can view the deployment descriptors and bindings in the administrative console.

Similar to Java API for XML-based RPC (JAX-RPC) Web services, you can use deployment descriptors to describe JAX-WS Web services. For JAX-WS Web services, the use of the webservices.xml deployment descriptor is optional because you can use annotations to specify all of the information that is contained within the deployment descriptor file. You can use the deployment descriptor file to augment or override existing JAX-WS annotations. Any information that you define in the webservices.xml deployment descriptor overrides any corresponding information that is specified by annotations.

1. Open the administrative console.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Manage Modules**.
   - Click **View Web services client deployment descriptor extension**.
   - Click **View Web services server deployment descriptor**.
   - Click **View Web services server deployment descriptor extension**.
3. Click **Expand All** to view the deployment descriptor contents.
4. Verify deployment descriptor and bindings configurations.

What to do next

You have viewed and verified the deployment descriptors and bindings for the Web services application.

Configuring the scope of a Web service port

When a Java API for XML-based RPC (JAX-RPC) Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains

deployment information for the Web module or enterprise bean module, including implementation scope, client bindings and deployment descriptor information. There are three levels of scope that can be set: application, session and request.

Before you begin

Deploy a Web service into the WebSphere Application Server. To learn more, read about deploying Web services applications onto application servers.

About this task

The Web Services for Java Platform, Enterprise Edition (Java EE) specification states that Web services implementations must be stateless. Therefore, to maintain specification compliance, the scope can remain at the application level because the state relevant to the individual sessions level or the requests level is not supposed to be maintained in the implementation. If you want to deviate from the specification and want to access a different JavaBean instance, because you are looking for information that is located in another JavaBean implementation, the scope settings need to change.

The setting that you configure for the scope determines how frequently a new instance of a service implementation class is created for the Web service ports in a module. Use this task to configure the scope of a Web service port.

This task applies only to Java API for XML-based RPC (JAX-RPC) Web services.

To change the scope setting in the administrative console:

1. Open the administrative console.
2. Click **Applications > Application Types > WebSphere enterprise applications *application_name* > Manage Modules > *module instance* > Web services implementation scope**
3. Set the scope to application, session or request. The application scope causes the same instance of the implementation to be used for all requests on the application. The session scope causes the same instance to be used for all requests in each session. The request scope causes a new instance to be used for every request. For example, with the scope set to application, every message that comes to the server accesses the same JavaBean instance because that is the way the scope settings are configured.
4. Click **Apply**.
5. Click **OK**.

Results

The scope for a Web service port is configured.

What to do next

Now you can finish any other configurations, start or stop the application, and verify the expected behavior of the Web service.

Web services implementation scope

Use this page to view and manage the scope of the ports of a Web service application.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Manage Modules > *module_instance* > Web services implementation scope**.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Related tasks

“Configuring the scope of a Web service port” on page 768
When a Java API for XML-based RPC (JAX-RPC) Web service application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise bean module, including implementation scope, client bindings and deployment descriptor information. There are three levels of scope that can be set: application, session and request.

Port:

Specifies a port name for a Web service. A module can contain one or more Web services, each of which can contain one or more ports.

Web service:

Specifies the name of the Web service. A module can contain one or more Web services.

URI:

Specifies the Uniform Resource Identifier (URI) of the binding file that defines the scope. The URI is relative to the Web module.

Scope:

Specifies the scope of a port. The valid values for scope are request, session and application.

The scope value determines when a new instance of a service implementation is created for the Web service ports in a module. When the scope value is set to application, the same instance of the implementation is used for all requests on the application. When the scope value is set to session, the same instance is used for all requests on each session. When the scope value is set to request, a new instance is created for every request.

Making deployed Web services applications available to clients

You can publish WSDL files to the file system. If you are a client developer or a system administrator, you can use WSDL files to enable clients to connect to Web services.

Before you begin

The publish WSDL administrative console panel supports both JAX-RPC and JAX-WS services. The publish WSDL panel generates a zip file that contains WSDL files for all modules in an application that contains JAX-WS or JAX-RPC Web services. Read about providing the HTTP endpoint URL information to learn how the URL information affects the content of the published WSDL.

To publish a Web Services Description Language (WSDL) file you need an enterprise application, also known as an enterprise archive (EAR) file, that contains a Web services-enabled module and has been deployed into WebSphere Application Server. To learn how to deploy Web services, see the documentation on deploying Web services onto application servers.

About this task

The purpose of publishing the WSDL file is to provide clients with a description of the Web service, including the URL identifying the location of the service.

After installing a Web services application, and optionally modifying the endpoint information, you might need WSDL files containing the updated endpoint informations to make deployed Web services application to be available to clients.

Before you publish a WSDL file, you can configure Web services to specify endpoint information in the form of URL fragments to enable full URL specification of WSDL ports. Refer to the tasks describing configuring endpoint URL information.

The WSDL files for each Web services-enabled module are published to the file system location you specify. You can provide these WSDL files to clients that want to invoke your Web services.

You can specify endpoint information for HTTP ports, for Java Message Service (JMS) ports, or you can directly access enterprise beans that are acting as Web services.

1. Configure the Web services client bindings.
2. Configure the URL endpoint information for HTTP bindings. Do one of the following depending on what kind of bindings you are using:
   - Configure the URL endpoint information for JMS bindings.
3. Externalize or publish the WSDL file out of the application. You can complete this task in the following ways:
   - Publish a WSDL file with the **wsadmin** command tool.

What to do next

Apply security to the Web service.

Configuring Web services client bindings

When a Web services application is deployed into WebSphere Application Server, an instance is created for each application or module. The instance contains deployment information for the Web module or enterprise JavaBean (EJB) module, including client bindings.

Before you begin

Deploy a Web service into your WebSphere Application Server instance. Read about deploying Web services applications onto application servers.

You must know the topology of the URL endpoint address of the Web services servers and which Web service the client depends upon. You can view the deployment descriptors in the administrative console to find the topology information. View Web services server deployment descriptors for more information.

About this task

The client bindings define the Web Services Description Language (WSDL) file name and preferred ports. The relative path of a Web service in a module is specified within a compatible WSDL file that contains the actual URL to be used for requests. The address is only needed if the original WSDL file did not contain a URL, or when a different address is needed. For a service endpoint with multiple ports, you need to define an alternative WSDL file name.

The following steps describe how to edit bindings for a Web service after these bindings are deployed on a server. When one Web service communicates with another Web service, you must configure the client bindings to access the downstream Web service.

To configure client bindings through the administrative console:

1. Open the administrative console.

2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Manage Modules** > *module_instance* > **Web services client bindings**.
3. Find the Web service you want to update.

   The Web services are listed in the **Web Service** field.
4. Select the WSDL file name from the drop down box in the WSDL file name field.
5. Click **Edit** in the Preferred port mappings field to configure the default port to use.
   a. Specify the port type and the preferred ports in the Port type and Preferred ports fields.

      Configuring the preferred port enables you to select an optimal port implementation use non-SOAP protocols. See RMI-IIOP Web services using JAX-RPC for more information about using non-SOAP protocols.
   b. Click **Apply** and **OK**.
6. Click **Edit** in the Port information field to configure the request timeout, the overridden endpoint, and the overridden binding namespace for a port.

   Configuring the request timeout accommodates complex topologies that can have multiple cascaded Web services that involve multiple hops or long-running services.

   You can configure Timeout values based on observed behavior of the overall system as integration proceeds. For example, a Web service client might time out because of changing network conditions or the performance of an external Web service. When you have applications containing Web services clients that timeout, you can change the request time out values for the clients.

   You can specify an endpoint URL to override the current endpoint. A client invoking a request on this port uses this endpoint instead of the endpoint specified in the WSDL file. You can specify the **Overridden endpoint URL** value for both Java API for XML-Based Web Services (JAX-WS) clients and Java API for XML-based RPC (JAX-RPC) clients.

   Note: The **Overridden endpoint URL** field is applicable for both JAX-WS and JAX-RPC clients. The other fields on this administrative console page are only applicable for JAX-RPC clients.
   a. Click **Apply** and **OK**.

Results

Your Web service client bindings are configured.

What to do next

Now you can finish any other configurations, start or restart the application, and verify the expected behavior of the Web service.

Web services client bindings

The client bindings define the Web Service Description Language (WSDL) file name, preferred ports and other port information. Use this page to specify the client bindings and the port mappings for the Web services in a module.

A Web service can specify the relative path within the module of a compatible WSDL file containing the actual URL to be used for requests. The relative path only needs to be specified if the original WSDL file does not contain a URL or when a different URL is needed. For a service endpoint with multiple ports defined, a preferred mapping specifies the default port to use for a port type.

To view this administrative console page, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications > *application_name*** .
2. Click **Manage modules > *URI_file_name* > Web services client bindings** .

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

Web service:

Identifies the name of this Web service. A module can contain one or more Web services.

EJB:

Identifies the name of the EJB for the EJB modules.

WSDL file name:

Specifies the WSDL file name, which is relative to the module. Locate the WSDL file name in the drop down menu.

Preferred port mappings:

Specifies and manages the preferred port type mapping for a Web service when a particular port type is requested.

Click **Edit** to edit the preferred port mapping information on the **Preferred port mappings** panel.

Port information:

Specifies additional configuration information for the ports of this Web service.

Click **Edit** to edit the port information on the **Port information** panel. You can set a request timeout, override an endpoint and override a binding namespace for each client port.

Preferred port mappings:

Use this page to view and manage a preferred portType mapping for a Web service.

This administrative console panel applies only to Java API for XML-based RPC (JAX-RPC) applications.

When you have multiple ports that reference the same portType (service endpoint interface), a preferred port specifies the port to use when the `Service.getPort(Class SEI)` method is called with only the service endpoint interface.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications *application_name* > Manage Modules > *module instance* > Web services client bindings > Edit > *preferred_port_instance***.

portType:

Specifies the portType.

The preferred port and the portType values are both of the type `java.xml.namespace.QName`.

Preferred port:

Specifies the preferred port to be associated with a particular portType. The `Service.getPort(Class)` method returns the preferred port associated with the specified service endpoint interface class (portType).

The preferred ports available are listed, as well as a value of `None`, which indicates no preferred port is selected.

Web services client port information:

Use this page to specify a request timeout, override an endpoint, and override a binding namespace for a Web services client port.

A Web service can have multiple ports. You can view and configure the port attributes for each defined Web service port. The Web services are listed on the Web services client bindings panel.

To view this page, click **Applications > Application Types > WebSphere enterprise applications *application_name* > Manage Modules > *module_instance* > Web services client bindings > Edit** .

This administrative console panel applies to both Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services. The **Overridden endpoint URL** field is the only field supported for JAX-WS clients. The other fields are not applicable for JAX-WS clients.

Port:

Specifies the name of a port.

Request timeout:

Specifies the time, in seconds, that a Web service client waits for a request to complete on this port. If a timeout is not specified, the default request timeout for the client to wait is `360` seconds. If the value is set at `0` (zero), the client's request does not timeout. This field is supported only for JAX-RPC clients.

A typical use for this setting is to customize the client's behavior when it is configured to use a JMS transport to access a Web service to make it wait longer for an expected completion. Depending upon network conditions, or the nature of a Web service implementation, it might be necessary to tune the timeout.

Overridden endpoint URL:

Specifies the name of an endpoint that is used to override the current endpoint. A client invoking a request on this port uses this endpoint instead of the endpoint specified in the WSDL file. This field is supported for both JAX-WS and JAX-RPC clients.

If an assembled application contains a Web service client that is statically bound, the client is locked into using the implementation (service end point) identified in the WSDL file used during development. Overriding the endpoint is an alternative to configuring the deployed WSDL attribute.

The overridden endpoint URI attribute is specified on a per port basis. It does not require an alternative WSDL file within the module. The overridden endpoint URI takes precedence over the deployed WSDL attribute. The client uses this value for the service end point URI or SOAP address, instead of the value in the static client bindings.

Overridden binding:

Specifies the WSDL file binding namespace URI to use with this port, instead of the namespace in the WSDL file. This binding does not need to exist in the WSDL file. A client invoking a request on this port uses this binding instead of the binding specified in the WSDL file. An overridden binding namespace cannot be specified unless an overridden endpoint is specified. This field is supported only for JAX-RPC clients.

Configuring endpoint URL information for HTTP bindings

Configuring a service endpoint is necessary to connect Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services clients to any Web services among the components being assembled or to any external Web services.

Before you begin

You can develop an HTTP accessible Java API for XML-based remote procedure call (JAX-RPC) or Java API for XML Web Services (JAX-WS) Web service when you already have a JavaBean object to enable as a Web service. For additional information, see the using HTTP to transport Web services requests documentation.

About this task

You can specify HTTP URL prefixes for Web services that are accessed through HTTP by using the Provide HTTP endpoint URL information panel in the administrative console. The HTTP URL prefixes provide location specific information and are used to form complete endpoint URLs that are included within published WSDL files.

Note: The Provide HTTP panel in the administrative console displays modules that contain Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services. You can use the Provide HTTP panel to provide URL information for both types of Web services, however, the panel does not indicate which type of service that you are working with.

To configure these prefixes with the administrative console:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Provide HTTP endpoint URL information**.
3. Specify the URL prefixes for the Web service.

   In this step you specify the protocol (HTTP or HTTPS), as well as the *host_name* and *port_number* used in the endpoint URL. You can select a prefix from a predefined list, by selecting the default HTTP URL prefix, or you can use a custom HTTP URL prefix.

   a. Select **Default HTTP URL prefix** or **Custom HTTP URL Prefix**.

      If you select the default HTTP URL prefix, a list provides you with a choice of endpoint URL prefixes. The list is a combination of two sets of ports in the module: the virtual host ports and the application server ports. Use a prefix from this list if the application server of the Web service is accessed directly. Select a value and also select the check box of the modules to use the prefix.

      If you want to use a custom HTTP URL prefix, type the value in the field. Select the check box to use in the prefix.

      If you configure a custom HTTP URL prefix, , you must also configure the custom JVM property, `com.ibm.ws.webservices.enableHTTPPrefix` in the administrative console and set the value to `true`. You must restart the application server after this custom property has been defined so that this property is used by the system. Setting this custom JVM property is required so the custom HTTP endpoint prefix information is correctly displayed in the ?WSDL query that is returned from the browser and the URL field of the WSDL file that is returned to the client. If this custom property is not defined with the value of `true`, the custom HTTP URL prefix is not reflected in the WSDL file that the service returns to the client. To learn how to configure this custom JVM property, see the documentation on configuring additional HTTP transport properties using the JVM custom property panel in the administrative console.

   b. Click **Apply**.

      The URL prefix, whether default or custom, is copied to the selected module HTTP URL prefix field.

   c. Click **OK**. The URL information is saved to your workspace.

Results

You have specified the partial URL information that is used to form the target endpoint addresses in the WSDL files that are published using the Publish WSDL files panel.

What to do next

Configure any other URL endpoint information for Java Message Service (JMS) bindings and direct Enterprise JavaBeans (EJB) access. Then publish the WSDL files to make the deployed Web services application available to clients.

Provide HTTP endpoint URL information

Use this page to specify endpoint URL prefix information for Web services accessed by HTTP. Prefixes are used to form complete endpoint addresses included in published Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Provide HTTP endpoint URL information**.

You can specify a portion of the endpoint URL to be used in each Web service module. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's `soap:address` element.

This administrative console panel applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

Specify endpoint URL prefixes for Web services:

Specifies the *protocol* (either http or https), *host_name*, and *port_number* to be used in the endpoint URL.

You can select a prefix from a predefined list using the **HTTP URL prefix** or **Custom HTTP URL prefix** field.

The URL prefix format is *protocol://host_name:port_number*, for example, http://*myHost*:9045. The actual endpoint URL that is contained in a published WSDL file consists of the prefix followed by the module's context-root and the Web service url-pattern, for example, http://*myHost*:9045/services/*myService*.

Select default HTTP URL prefix:

Specifies the drop down list associated with a default list of URL prefixes. This list is the intersection of the set of ports for the module's virtual host and the set of ports for the module's application server. Use items from this list if the Web services application server is accessed directly.

To set an HTTP endpoint URL prefix, select **Select default HTTP URL prefix** and select a value from the drop down list. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP URL prefix** field of any module whose check box is selected.

Select custom HTTP URL prefix:

Specifies the *protocol*, *host*, and *port_number* of the intermediate service if the Web services in a module are accessed through an intermediate node, for example the Web services gateway or an IHS server.

To set a custom HTTP endpoint URL prefix, you must also configure the custom JVM property, `com.ibm.ws.webservices.enableHTTPPrefix` in the administrative console and set the value to `true`. Setting this custom JVM property is required so the custom HTTP URL is correctly populated in the URL field of the WSDL file that is returned to the client. If this custom JVM property is not configured, the custom HTTP URL prefix is not in the URL field in the copy of the WSDL file that the service returns to the client. To learn how to configure this custom JVM property, see the documentation on configuring additional

HTTP transport properties using the JVM custom property panel in the administrative console. You must restart the application server after this custom property has been defined so that this property is used by the system.

After the `com.ibm.ws.webservices.enableHTTPPrefix` custom JVM property is configured, select **Select custom HTTP URL prefix** and enter a value. Select the check box of the modules that are to use the prefix and click **Apply**. When you click **Apply**, the entry in the **Select default HTTP URL prefix** or **Select custom HTTP URL prefix** fields, depending on which is selected, is copied into the **HTTP endpoint URL prefix** field of any module whose check box is selected.

Configuring endpoint URL information for JMS bindings

WebSphere Application Server supports the use of the Java Message Service (JMS) API to transport Web services requests, as an alternative to using HTTP.

Before you begin

The application server supports use of the Java Message Service (JMS) API to transport Web services requests, as an alternative to HTTP transport. Read about using the Java Message Service (JMS) to transport Web services requests to learn more about how Web service clients and servers can communicate through JMS queues and topics instead of through HTTP connections.

About this task

Configuring a service endpoint is necessary to connect Web service clients to any Web services among the components being assembled or to any external Web services. You can configure the endpoint URL information for JMS during application installation

In this task, enter the JMS endpoint URL prefix to use for each Web service-enabled Enterprise JavaBeans (EJB) Java archive (JAR) file that belong to the application. The JMS endpoint URLs are included in the Web Service Description Language (WSDL) files published for clients to use.

You can specify HTTP URL prefixes for Web services that are accessed through HTTP by using the Provide HTTP endpoint URL information panel in the administrative console. These prefixes are used to form complete endpoint addresses that are included in WSDL files when published.

You can specify JMS URL prefixes by using the Provide JMS and EJB endpoint URL information panel in the administrative console during or after application installation.

This task applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

To configure JMS URL prefixes:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Provide JMS and EJB endpoint URL information**.
3. Locate the list of Web services modules that are accessible through JMS transport.
4. Type the JMS URL fragment in the **URL fragment** field. Enter a URL fragment that is a prefix to the initial URL part that is obtained by examining the deployment information of the Web service. See the usage scenario following this task for more information.

   The value that you enter is used to define the location attribute of the port soap:address element within the WSDL file that is published using the *application_name*`_ExtendedWSDLFiles.zip` or the *application_name*`_WSDLFiles.zip` file on the Publish WSDL zip files panel.

Results

You have a Web service that is accessible through the JMS transport and configured with JMS bindings.

Example

Suppose an application called StockQuoteService contains an EJB JAR file that is named StockQuoteEJB, which contains one or more Web services that are accessible through the JMS transport. In Using SOAP over Java Message Service to transport Web services requests, you defined a queue with the Java Naming and Directory Interface (JNDI) name of `jms/StockQuote_Q`, and a connection factory with the JNDI name of `jms/StockQuote_CF`, for your application. In this example, you specify the following string as the JMS URL prefix within the Provide JMS and EJB endpoint URL information panel:

```
jms:/queue?destination=jms/StockQuote_Q&connectionFactory=jms/StockQuote_CF
```

The WSDL publisher uses this partial URL string to produce the actual JMS URL for each port component that is defined in the module. The `targetService=<port_name>` string is added to the end of the JMS URL, for example:

```
jms:/queue?destination=jms/StockQuote_Q&connectionFactory=jms/StockQuote_CF&targetService=getQuote
```

The published WSDL file is used by clients to invoke the Web service.

What to do next

Publish the WSDL files to make the deployed Web services application available to clients.

Provide JMS and EJB endpoint URL information

Use this page to specify endpoint URL fragments for Web services accessed through SOAP and Java Message Service (JMS) or directly as enterprise beans. Fragments are used to form complete endpoint addresses included in published Web Services Description Language (WSDL) files.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Provide JMS and EJB endpoint URL information**.

You can specify a fragment of the endpoint URL to be used in each Web service module. In a published WSDL file, the URL defining the target endpoint address is found in the location attribute of the port's `soap:address` element.

If you are using Web services modules that are configured to use JMS or configured to access enterprise beans directly, these modules are listed on this panel.

This administrative console panel applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

URL fragment for JMS:

Specifies a URL fragment for Web services accessed through a JMS transport. You can enter a value that is used to define the `soap:address` of a Web service. When WSDL files are published, a URL is formed using this fragment and is contained in the WSDL files.

The URL fragment that is entered as a value is a prefix to which the targetService=property is appended to form a complete JMS URL endpoint. The default value is obtained by examining the installed service's deployment information, for example, `jms:/queue?destination=jms/MyQueue&connectionFactory=jms/MyCF`.

This information is obtained from the Web service's configured JMS endpoint, which is a Message Driven Bean (MDB) defined by the **endpointEnabler** command-line tool. You can modify the URL fragment, for

example, by adding properties. The URL fragment is combined with the targetService property to form the complete URL, for example, `jms:/queue?destination=jms/MyQueue&connectionFactory=jms/MyCF &priority=5&targetService=GetQuote`.

URL fragment for EJB:

Specifies a URL fragment for Web services accessed through an EJB binding. You can enter a value used to define the location attribute of the port's `generic:address` element of a Web service. This port address is contained in the WSDL zip file when the zip file is published using the ***application_name_ExtendedWSDLFiles*.zip** field on the **Publish WSDL zip file** panel.

The URL fragment value entered is a suffix, which is appended to the initial part of the URL obtained by examining the Web service's deployment information. For example, the following URL fragment can be obtained from the EJB's deployment information: `wsejb:/ com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome`.

In this case, you can enter the following information in the URL fragment field, `jndiProviderURL=corbaloc:iiop:myhost.mycompany.com:2809`, which results in this endpoint URL, `wsejb:/com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome &jndiProviderURL=corbaloc:iiop:myhost.mycompany.com:2809`.

Configuring endpoint URL information to directly access enterprise beans

WebSphere Application Server supports directly accessing an enterprise bean as a Web service, as an alternative to using HTTP or Java Message Service (JMS) to transport requests between the server and the client. The EJB module that is used as a Web service contains a Web Services Description Language (WSDL) file that contains EJB bindings.

Before you begin

To learn more about the process of directly accessing an enterprise bean as a Web service, review the topic Using EJB bindings to invoke an enterprise bean from a Web services client .

About this task

Configuring a service endpoint is necessary to connect Web service clients to any Web services among the components being assembled or to any external Web services.

You can specify Web address endpoints of the enterprise bean for Web services that are accessed directly by EJB bindings using the Provide JMS and EJB endpoint Web address information panel in the administrative console.

If you have modules that are configured for using direct EJB access, the modules are listed on the Provide JMS and EJB endpoint Web address information panel in the administrative console. The EJB endpoint is only available in the WSDL that is found in the *application_name*_ExtendedWSDLfiles.zip file.

You can specify a fragment of the endpoint Web address for the Web services in each module.

To configure the Web address endpoints of the enterprise bean with the administrative console:

1. Open the administrative console.
2. Click **Applications** > **Enterprise Applications** > *application_instance* > **Provide JMS and EJB endpoint URL information**.
3. Locate the list of EJB modules.
4. Select the application module.

5. Type the Web address fragment in the **URL fragment** field.

   Enter a Web address fragment that is a suffix to the initial Web address part that is obtained by examining the Web service deployment information. See the example following this task for more information.

   The value that you enter is used to define the location attribute of the port generic:address element within the WSDL file that is published using the *application_name*_ExtendedWSDLFiles.zip file name link on the Publish WSDL zip files panel. The zip file names are listed as links on the panel.
6. Click **OK**.
7. Click **Save**.

Results

You have configured endpoints of the enterprise bean for Web services that are accessed directly by EJB bindings.

Example

The following example illustrates a Web address fragment to enter in the URL fragment field.

The following Web address information can be obtained from the deployment descriptor of an enterprise bean:

```
wsejb:/com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome
```

Enter the following Web address fragment in the URL fragment field:

```
jndiProviderURL=corbaloc:iiop:myhost.mycompany.com:2089
```

The results are shown in the following example:

```
wsejb:/com.acme.sample.MyStockQuoteHome?jndiName=ejb/MyStockQuoteHome&jndiProviderURL=
corbaloc:iiop:myhost.mycompany.com:2089
```

What to do next

Provide a description of the Web service to the service requestor by publishing WSDL files.

Publishing WSDL files using the administrative console

You can publish a Web Services Description Language (WSDL) file using the WebSphere Application Server administrative console.

Before you begin

Before completing this task, you need to install or deploy the Web service. After deployment, configure the URL endpoint tasks for your transport:

- Configure endpoint URL information for HTTP bindings.
- Configure endpoint URL information for JMS bindings.
- Configure endpoint URL information to directly access enterprise beans

About this task

By publishing a WSDL file, you are providing clients with a description of the Web service, including the URL identifying the location of the service.

The WSDL files in each Web services-enabled module are published to the file system location you specify. You can provide these WSDL files in the development and configuration process of the Web service clients so they can invoke your Web services.

This task applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

Review Publishing WSDL files for more ways to publish WSDL files.

To publish an application's WSDL file with the administrative console:

1. Open the administrative console.
2. Click **Applications > Application Types > WebSphere enterprise applications > *application_name***.
3. Under Web Services Properties, click **Publish WSDL files**. This takes you to the **Publish WSDL zip files** page.
4. Click the WSDL zip file to download. The zip file contains the application's published WSDL files. The zip file `Extended`*`WDLFiles`*`.zip` contains EJB binding information. It can also contain JMS or HTTP binding information. The zip file *`WSDLFiles`*`.zip` only contains JMS or HTTP binding information.

What to do next

Apply security to your Web service.

Publish WSDL zip files settings

Use this page to publish Web Services Description Language (WSDL) files.

This administrative console panel applies for Java API for XML-Based Web Services (JAX-WS) and Java API for XML-based RPC (JAX-RPC) Web services.

The publish WSDL panel generates a zip file that contains WSDL files for all modules in an application that contains a JAX-WS or JAX-RPC Web service. Read about providing the HTTP endpoint URL information to learn how the URL information affects the content of the published WSDL.

To view this administrative console page, click **Applications > Application Types > WebSphere enterprise applications > *application_name* > Publish WSDL files**.

When you click **OK**, a panel showing one or several zip file names displays. Each zip file contains a WSDL file that represents the Web services-enabled modules in the application. When you select a zip file to publish, a dialogue displays from which you can choose where to create the zip file. Within the published zip files, the directory structure is *application_name/module_name/[META-INF|WEB-INF]/wsdl/wsdl_file_name*.

In a published WSDL file, the location attribute of a port's `soap:address` element contains the endpoint URL through which the Web service is accessed. Using the **Provide HTTP endpoint URL information** and the **Provide JMS and EJB endpoint URL information** panels, configure the endpoint URLs to be used for the Web services in each module.

*application_name*_***WSDLFiles.zip:***

Specifies the *`application_name`*`_WSDLFiles.zip` file containing the WSDL that describes Web services that are accessible by standard SOAP-based ports.

*application_name*_***ExtendedWSDLFiles.zip:***

Specifies the *application_name*_ExtendedWSDLFiles.zip file containing the WSDL file that describes the Web services available, including SOAP-based and non-SOAP based (for example, EJB) ports.

If there are no Web services configured for direct EJB access, this zip file name is not displayed. Do not use this zip file if you want to produce a WSDL file compliant to standards.

Publishing WSDL files using a URL

You can publish a Web Services Description Language (WSDL) file using a URL.

Before you begin

Before you can publish a WSDL file using a URL, ensure the Web services-enabled application is installed and running.

The files referenced by the <wsdl-file> element in the webservices.xml might import other WSDL or XML Schema Definition (XSD) files. Typically, all WSDL or XSD files are initially placed into the META-INF/wsdl directory when using Enterprise JavaBeans (EJB) or the WEB-INF/wsdl directory when using JavaBeans. If your WSDL or XSD files are not placed in one of these directories, the file referenced by the <wsdl-file> and its imported files are copied to the wsdl directory for publishing purposes.

There are two different forms of URL query strings. The first appends /wsdl to the service and returns only HTTP and JMS bindings. The second appends /extwsdl to the service and returns the extended WSDL file, including HTTP, JMS, and EJB bindings. If a WSDL file contains only EJB bindings and the /wsdl query is used, an error message displays in the browser saying there are no HTTP or JMS bindings in the WSDL file. The error message suggests using the /extwsdl query instead. Publishing a WSDL file using a URL requires that the application have a Web module; either provided by the application or in the form of an HTTP router module. If an EJB application contains a WSDL file with only JMS or EJB Web service bindings, the **endptEnabler** command can be used to add an HTTP router module to the application.

Note: Only HTTP URLs are supported for publishing.

About this task

To publish a WSDL file using a URL:

1. Retrieve the outer-most WSDL file. The outer-most WSDL file is the WSDL file defined by the <wsdl-file> element in the webservices.xml file.

   Each Web service has an endpoint address, like http://example.com/services/stockquote. You can retrieve the outer-most WSDL file (defined by the <wsdl-file> element within the webservices.xml file) by appending the string "/wsdl" or "/wsdl/" to the endpoint address, for example,http://example.com/services/stockquote/wsdl.
2. Retrieve the imported WSDL files. When the outer-most WSDL file imports other WSDL or XSD files, these imported files can be retrieved by appending the relative path to the URL, which is used to retrieve the outer-most WSDL file. This is also true for WSDL files that import other files. This process is similar to the use of relative hyperlinks in HTML documents. If an HTML document contains a hyperlink to other documents, the relative path is appended to create the URL to access the hyperlinked documents.

Example

Suppose you have an application with the following directory structure:

```
<module-root>/
 WEB-INF/
  webservices.xml    /* the <wsdl-file> element points to "WEB-INF/wsdl/fooImpl.wsdl"*/
  web.xml
  ibm-webservices-bnd.xml
```

```
wsdl/
 fooImpl.wsdl   /* imports foo.wsdl which is an interface wsdl */
 foo.wsdl       /* type definition for the interface */
```

If the SOAP address for the foo service is `http://examples.com:9080/services/foo`, the simple way to retrieve the foo service's outer-most WSDL is with the following form: `http://examples.com:9090/services/foo/wsdl` or `http://examples.com:9090/services/foo/wsdl/`. The URL is redirected to `http://examples.com:9090/services/foo/wsdl/fooImpl.wsdl`, where `fooImpl.wsdl` is the name of the outer-most WSDL file.

Since the `fooImpl.wsdl` file has the import `<import namespace="http://examples.com/foo" location="a/b/foo.wsdl>`, use the URL `http://examples.com:9090/services/foo/wsdl/a/b/foo.wsdl` to obtain the `foo.wsdl` file.

Creating a monitor for WebSphere Application Server for WSDM resources (deprecated)

The Web Services Distributed Management (WSDM) monitoring support provided in the Version 6.1 Feature Pack for Web Services uses an external management software. It is, however, more useful to build a custom WebSphere Application Server monitor for WSDM. Use this task to create a WSDM monitor that is based on the Java API for XML Web Services (JAX-WS) programming model.

Before you begin

Note: IBM WebSphere Application Server supports the Java API for XML-Based Web Services (JAX-WS) programming model and the Java API for XML-based RPC (JAX-RPC) programming model. JAX-WS is the next generation Web services programming model extending the foundation provided by the JAX-RPC programming model. Using the strategic JAX-WS programming model, development of Web services and clients is simplified through support of a standards-based annotations model. Although the JAX-RPC programming model and applications are still supported, take advantage of the easy-to-implement JAX-WS programming model to develop new Web services applications and clients.

About this task

Complete the following steps to create a WSDM monitor:

1. Retrieve the relative URLs. The following table contains a list of URLs that you can use to retrieve the Web Services Description Language (WSDL) files for each supported resource that is in the product.

Table 6. Relative URLs to WSDM WSDL files

Resource	URL
Service group	/websphere-management/services/service-group?wsdl
Application	/websphere-management/services/application?wsdl
Application server	/websphere-management/services/applicationserver?wsdl
Data source	/websphere-management/services/datasource?wsdl
Enterprise JavaBeans	/websphere-management/services/ejb?wsdl
Java virtual machine (JVM)	/websphere-management/services/jvm?wsdl
Servlet	/websphere-management/services/servlet?wsdl
WebSphere cluster	/websphere-management/services/websphere cluster?wsdl
WebSphere domain	/websphere-management/services/ webspheredomain?wsdl

Table 6. Relative URLs to WSDM WSDL files (continued)

Resource	URL
Web service	/websphere-management/services/webservices?wsdl

2. Create a skeleton code for the service endpoint interface (SEI) files for the Web services client. After retrieving the relative URLs, create a skeleton code for the SEI files by running the WSDL to Java tooling against the WSDM WSDL files. You can use the IBM Rational Application Developer or wsimport command utility for this task. Run the wsimport utility against the retrieved WSDL file.

   Note: At a minimum, WSDL files for the service group resource and the resource that you want must be compiled.

   When you finish this step, you have created the client code that is used to communicate with the service provider, which is the WSDM service endpoint.
3. Write the code for the JAX-WS WSDM client. Review the following JAX-WS WSDM client sample code before you start writing the client code. The sample client code demonstrates a complete implementation of a client suitable for polling statistics for a resource and is used to delegate the setting up of WS-Addressing resource references and request and response processing.

   Sample WSDM JAX-WS client

```
import java.net.URI;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.xml.bind.JAXBElement;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;

import org.oasis_open.docs.wsrf.rp_2.GetResourcePropertyResponse;
import org.oasis_open.docs.wsrf.sg_2.EntryType;
import org.oasis_open.docs.wsrf.sgw_2.ServiceGroupPortType;
import org.oasis_open.docs.wsrf.sgw_2.ServiceGroupService;
import org.w3c.dom.Element;

import com.ibm.wsspi.wsaddressing.AttributedURI;
import com.ibm.wsspi.wsaddressing.EndpointReference;
import com.ibm.wsspi.wsaddressing.EndpointReferenceManager;
import com.ibm.wsspi.wsaddressing.WSAConstants;
import com.ibm.wsspi.wsaddressing.WSAddressingFactory;
import com.ibm.xmlns.prod.websphere.management.servlet.ServletPortType;
import com.ibm.xmlns.prod.websphere.management.servlet.ServletService;

public class WsdmJaxWsClient {

 static QName WAS_MBEAN_ID = new QName("http://ibm.com/2006/v1.3", "WAS_Resource_MbeanIdentifier");
 static QName WAS_RESOURCE_ID = new QName("http://ibm.com/2006/v1.3", "WAS_Resource_MRID");
 static QName WAS_RESOURCE_TYPE = new QName("http://ibm.com/2006/v1.3", "WAS_Resource_Type");
 private String ServerAndHost = null;

 public WsdmJaxWsClient(String serverAndHost) {
  this.ServerAndHost = serverAndHost;
 }

 /**
  * Get a list of available resources from WSDM application
  */
 public List<EntryType> getResources() throws Exception {
  // Create service and obtain service reference
  ServiceGroupService sgs = new ServiceGroupService();
  ServiceGroupPortType sgp = sgs.getServiceGroupPort();

  // Create wsa:Action
  URI actionUri = new URI(
    "http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest");
  AttributedURI wsaAction = WSAddressingFactory
    .createAttributedURI(actionUri);

  // Create wsa:To and populate reference parameters
  String sgServicePortURI = "http://" + ServerAndHost
    + "/websphere-management/services/service-group";
  QName resourceId = new QName("http://ws.apache.org/muse/addressing",
    "ResourceId", "muse-wsa");
  EndpointReference wsaDestinationEpr = EndpointReferenceManager
    .createEndpointReference(new URI(sgServicePortURI));
  wsaDestinationEpr.setReferenceParameter(resourceId, "MuseResource-1");

  // Populate requestContext
```

```
 Map<String, Object> rc = ((BindingProvider) sgp).getRequestContext();
 rc.put(WSAConstants.WSADDRESSING_ACTION, wsaAction);
 rc.put(WSAConstants.WSADDRESSING_DESTINATION_EPR, wsaDestinationEpr);
 rc.put(BindingProvider.SOAPACTION_USE_PROPERTY, Boolean.TRUE);
 rc
   .put(
     BindingProvider.SOAPACTION_URI_PROPERTY,
     "http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest");
 rc.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, sgServicePortURI);

 // Retrieve a list of available resources
 QName entryQName = new QName("http://docs.oasis-open.org/wsrf/sg-2",
   "Entry");
 GetResourcePropertyResponse grpr = sgp.getResourceProperty(entryQName);

 // The returned list<object> must be converted to a more meaningful type
 List<Object> inputList = grpr.getAny();
 List<EntryType> outputList = new ArrayList<EntryType>(inputList.size());

 // Complete the element by element conversion
 for (Object obj : inputList) {
  JAXBElement jbe = (JAXBElement) obj;
  EntryType entry = (EntryType) jbe.getValue();

  outputList.add(entry);
 }

 return outputList;
}

/**
 * Read a property of specified entry
 */
public Object getResourceProperty(EntryType entry, QName property)
  throws Exception {
 // Create service and obtain service reference
 ServletService rs = new ServletService();
 ServletPortType rp = rs.getServletPort();

 // Create wsa:Action
 URI actionUri = new URI(
   "http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest");
 AttributedURI wsaAction = WSAddressingFactory
   .createAttributedURI(actionUri);

 // Create wsa:To and populate resource parameters
 String servicePortURI = getResourceAddress(entry);
 // QName resourceId = new QName("http://ws.apache.org/muse/addressing",
 // "ResourceId", "muse-wsa");
 EndpointReference wsaDestinationEpr = EndpointReferenceManager
   .createEndpointReference(new URI(servicePortURI));
 wsaDestinationEpr.setReferenceParameter(WAS_RESOURCE_ID,
   findListMember(entry, WAS_RESOURCE_ID));
 wsaDestinationEpr.setReferenceParameter(WAS_MBEAN_ID, findListMember(
   entry, WAS_MBEAN_ID));
 wsaDestinationEpr.setReferenceParameter(WAS_RESOURCE_TYPE,
   findListMember(entry, WAS_RESOURCE_TYPE));

 // Populate requestContext
 Map<String, Object> rc = ((BindingProvider) rp).getRequestContext();
 rc.put(WSAConstants.WSADDRESSING_ACTION, wsaAction);
 rc.put(WSAConstants.WSADDRESSING_DESTINATION_EPR, wsaDestinationEpr);
 rc.put(BindingProvider.SOAPACTION_USE_PROPERTY, Boolean.TRUE);
 rc
   .put(
     BindingProvider.SOAPACTION_URI_PROPERTY,
     "http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest");
 rc.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, servicePortURI);

 // Retrieve a list of available resources
 GetResourcePropertyResponse response = null;
 response = rp.getResourceProperty(property);

 // Return the property
 if (response != null) {
  return ((JAXBElement) response.getAny().get(0)).getValue();
 } else {
  return null;
 }
}

public String getResourceMBeanID(EntryType entry) {
 return findListMember(entry, WAS_MBEAN_ID);
}

public String getResourceMRID(EntryType entry) {
 return findListMember(entry, WAS_RESOURCE_ID);
}

public String getResourceType(EntryType entry) {
```

```
  return findListMember(entry, WAS_RESOURCE_TYPE);
 }

 /**
  * Extract the ERPs address
  */
 public String getResourceAddress(EntryType entry) {
  if (entry == null)
   return null;
  else
   return entry.getMemberServiceEPR().getAddress().getValue();
 }

 /**
  * EPRs cannot be looked up by their QName. This method does a linear search
  * of the EPR to find a particular reference parameter
  */
 private String findListMember(EntryType entry, QName name) {

  if (name == null || entry == null) return null;

  List<Object> res = entry.getMemberServiceEPR().getReferenceParameters()
    .getAny();

  for (Object ob : res) {
   Element e = (Element) ob;
   if (name.getLocalPart().equals(e.getNodeName()))
    return e.getTextContent();
  }

  return null;
 }
}
```

4. Create the WSDM monitor. The following example is used for polling a servlet for the number of requests that the servlet it is responding to.

   WSDM monitor

```
// Instantiate the client and instruct it to poll
// the server at localhost:9080
WsdmJaxWsClient wjc = new WsdmJaxWsClient("localhost:9080");

// Get a list of all available resources
List<EntryType> entries = wjc.getResources();

// Iterate through available resources until you find
// a servlet resource
EntryType servlet = null;
for (EntryType entry: entries){
 if ("servlet".equals(wjc.getResourceType(entry))){
  servlet = entry;
  System.out.println("URL " + wjc.getResourceAddress(entry));
  System.out.println("Type " + wjc.getResourceType(entry));
  System.out.println("MbeanID" + wjc.getResourceMBeanID(entry));
  System.out.println("ResourceID " + wjc.getResourceMRID(entry));
  break;
 }
}

// Request Concurrent Requests PMI statistics from this servlet
Object servletConcurrentRequests =
 wjc.getResourceProperty(servlet, J2EEConstants.CONCURRENT_REQUESTS_QNAME);

System.out.println(servletConcurrentRequests);
```

5. Obtain a list of available resources. The sample JAX-WS WSDM client provided in code example 1 shows a list of available resources. The list is filtered to obtain some resources and statistics from these resource objects. In this step, the client is initialized for an application server that is installed on a local machine and it is servicing HTTP requests on port 9080.

   Initialize WSDM client

```
// Instantiate the client and instruct it to poll
// the server at localhost:9080
WsdmJaxWsClient wjc = new WsdmJaxWsClient("localhost:9080");

// Get a list of all available resources
List<EntryType> entries = wjc.getResources();
```

6. Filter the available resources. The list of resources returned by getResources call is lengthy. This call returns a list of all resources that are currently available, not necessarily the particular resource type that you might be interested in.

   Filter resources

```
// Iterate through available resources until you find a servlet resource
EntryType servlet = null;
for (EntryType entry: entries){
 if ("servlet".equals(wjc.getResourceType(entry))){
  servlet = entry;
  System.out.println("URL " + wjc.getResourceAddress(entry));
  System.out.println("Type " + wjc.getResourceType(entry));
  System.out.println("MbeanID" + wjc.getResourceMBeanID(entry));
  System.out.println("ResourceID " + wjc.getResourceMRID(entry));
  break;
 }
}
```

You might want to expand this example to find a particular resource instead of finding the first available servlet. To complete this action, you need to do further filtering based on getResourceMBeanID. The string representing the resource is similar to the following code example:

Sample resource MBean identifier string

```
WebSphere:WebModule=WSDMDemo.war,name=myServlet,process=server1,Application=wsdm-demo,platform=dynamicproxy,
J2EEApplication=wsdmdDemo,node=demoNode01,J2EEName=wsdm-demo#WSDMDemo.war#mySelvlet,j2eeType=Servlet,
J2EEServer=server1,Server=server1,version=6.1.0.7,type=Servlet,mbeanIdentifier=
wsdm-demo#WSDMDemo.war#myServlet,cell=demoCell01,spec=1.0
```

7. Poll resources for statistics. Finally, you can now request the resource property. A list of available URLs for a given resource can be obtained from the WSDL file in Table 1. The following example is a list of properties for a servlet:

   Available properties for a servlet resource

```
<xsd:element name="ServletResourceProperties">
xsd:complexType>
 xsd:sequence>

 !--  WSDM MUWS Part 1 - Identity -->
 <xsd:element ref="muws1:ResourceId"></xsd:element>

 !--  WSDM MUWS Part 1 - ManageabilityCharacteristics -->
 <xsd:element ref="muws1:ManageabilityCapability" minOccurs="0" maxOccurs="unbounded"></xsd:element>

 !--  WSDM MUWS Part 2 - Description -->
 <xsd:element ref="muws2:Caption" minOccurs="0" maxOccurs="unbounded"></xsd:element>
 <xsd:element ref="muws2:Description" minOccurs="0" maxOccurs="unbounded"></xsd:element>
 <xsd:element ref="muws2:Version" minOccurs="0"></xsd:element>

 !--  WSDM MUWS Part 2 - Metrics -->
 <xsd:element ref="muws2:CurrentTime"></xsd:element>

 !--  J2EE Servlet -->
 <xsd:element ref="servlet:ConcurrentRequests"></xsd:element>
 <xsd:element ref="servlet:TotalRequests"></xsd:element>
 <xsd:element ref="servlet:NumberOfErrors"></xsd:element>
 <xsd:element ref="servlet:ResponseTime"></xsd:element>


 !--  J2EE ManagedObject -->
 <xsd:element ref="j2ee:ObjectName"></xsd:element>
 <xsd:element ref="j2ee:StateManageable"></xsd:element>
 <xsd:element ref="j2ee:EventProvider"></xsd:element>
 <xsd:element ref="j2ee:StatisticsProvider"></xsd:element>

 </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

8. Obtain statistics. You can use the following code example to poll for the concurrent requests property. Then, you can resolve the property to a QName and pass the property onto the client.

   Obtain statistics

```
// Request Concurrent Requests PMI statistics from this servlet
Object servletConcurrentRequests =
 wjc.getResourceProperty(servlet, new QName("http://www.ibm.com/xmlns/prod/websphere/management/j2ee/servlet", "ConcurrentRequests"));

System.out.println(servletConcurrentRequests);
```

Results

You have created a custom WSDM monitor that you can use to manage and control resources.

Use the code examples to help you in creating your custom monitor, including the SOAP conversation code example.

In addition to these code examples, there is a built-in code sample in the product that you can use to retrieve the statistics for a servlet resource. To execute this sample, run `java –classpath wsdm-demo.jar;<install_root>/runtimes/com.ibm.jaxws.thinclient_7.0.0.jar samples.wsdm.servlet.WsdmDemo`.

Example

SOAP conversation example

Requesting available resources

The following code example demonstrates how to request for all available resources.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Header>
    <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">http://test.ibm.com:8080/websphere-management/services/service-group</wsa:To>
  <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
       http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest</wsa:Action>
       <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:2f1abf03-ef3e-0837-b8e6-d34dc8e352e2</wsa:MessageID>
       <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing"><wsa:Address>
           http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address></wsa:From>
<muse-wsa:ResourceId xmlns:muse-wsa="http://ws.apache.org/muse/addressing" xmlns:wsa="http://www.w3.org/2005/08/addressing" wsa:IsReferenceParameter=
"true">MuseResource-1</muse-wsa:ResourceId>
</soap:Header>
 <soap:Body>
 <wsrf-rp:GetResourceProperty xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2" xmlns:wsrf-sg=
"http://docs.oasis-open.org/wsrf/sg-2">wsrf-sg:Entry</wsrf-rp:GetResourceProperty>
</soap:Body>
</soap:Envelope>
```

Available resources

This example is a response that has two available resources that are represented as wsrf-sg:Entry XML elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:soapenc="http:www.w3.org/2003/05/soap-encoding"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <env:Header>
  <wsa:To>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:To>
  <wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyResponse</wsa:Action>
  <wsa:MessageID>uuid:4eda30ea-c8ca-2998-2c9f-e2519175a1aa</wsa:MessageID>
  <wsa:RelatesTo RelationshipType="wsa:Reply">uuid:2f1abf03-ef3e-0837-b8e6-d34dc8e352e2</wsa:RelatesTo>
  <wsa:From>
   <wsa:Address>http://test.ibm.com:8080/websphere-management/services/service-group</wsa:Address>
   <wsa:ReferenceParameters><muse-wsa:ResourceId xmlns:muse-wsa="http://ws.apache.org/muse/addressing" IsReferenceParameter="true">
    MuseResource-1</muse-wsa:ResourceId></was:From>
   <was:RelatesTo>uuid:2f1abf03-ef3e-0837-b8e6-d34dc8e352e2</was:RelatesTo>

</env:Header>
<env:Body>
 <wsrf-rp:GetResourcePropertyResponse xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
 <wsrf-rp:Entry xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2"
 <wsrf-sg:ServiceGroupEntryEPR
 <wsa:Address> "http://test.ibm.com:8080/websphere-management/services/service-group-entry</wsa:Address>
 <wsa:ReferenceParameters><muse-wsa:ResourceId xmlns:muse-wsa="http://ws.apache.org/muse/addressing">MuseResource-103</muse-wsa:ResourceId>
 </wsa:ReferenceParameters>
 </wsrf-sg:ServiceGroupEntryEPR>
 <wsrf-sg:MemberServiceEPR>
 <wsa:Address>http://test.ibm.com:8080/websphere-management/services/ejb</wsa:Address>
 <wsa:ReferenceParameters>
      <was-wsdm:WAS_Resource_Type>
      <was-wsdm:WAS_Resource_ManagedNodeID xmlns:was-wsdm="http://ibm.com/2006/v1.3"
      <was-wsdm:WAS_Resource_Type xmlns:was-wsdm="http://ibm.com/2006/v1.3">ejb     <was-wsdm:WAS_Resource_MbeanIdentifier
 xmlns:was-wsdm="http://ibm.com/2006/v1.3">
WebSphere:name=EjbClient,process=server1,Application=jms-client,platform=dynamicproxy,J2EEApplication=jms-client,node=
Node01,J2EEName=jms-client#jms-ejb-client.jar#EjbClient,j2eeType=StatelessSessionBean,J2EEServer=server1,Server=server1,version=
7.0.0.0,type=StatelessSessionBean,mbeanIdentifier=jms-client#jms-ejb-client.jar#EjbClient,EJBModule=jms-ejb-client.jar,cell=
Cell01,spec=1.0</was-wsdm:WAS_Resource_MbeanIdentifier>
      <was-wsdm:WAS_Resource_MRID xmlns:was-wsdm="http://ibm.com/2006/v1.3">
ejb:WebSphere:name=EjbClient,process=server1,Application=jms-client,platform=dynamicproxy,J2EEApplication=jms-client,node=
Node01,J2EEName=jms-client#jms-ejb-client.jar#EjbClient,j2eeType=StatelessSessionBean,J2EEServer=server1,Server=server1,version=
7.0.0.0,type=StatelessSessionBean,mbeanIdentifier=jms-client#jms-ejb-client.jar#EjbClient,EJBModule=jms-ejb-client.jar,cell=
```

```
Cell01,spec=1.0</was-wsdm:WAS_Resource_MRID>
    </wsa:ReferenceParameters>
    </wsrf-sg:MemberServiceEPR>
    <wsrf-sg:Content/>
    </wsrf-sg:Entry>
    <wsrf-sg:Entry xmlns:wsrf-sg="http://docs.oasis-open.org/wsrf/sg-2">
    <wsrf-sg:ServiceGroupEntryEPR><wsa:Address>http://test.ibm.com:8080/websphere-management/services/service-group-entry</wsa:Address>
    <wsa:ReferenceParameters><muse-wsa:ResourceId xmlns:mus-wsa="http://ws.apache.org/muse/addressing"<MuseResource-97</muse-wsa:
ResourceId></wsa:ReferenceParameters>
    </wsrf-sg:ServiceGroupEntryEPR>
    <wsrf-sg:MemberServiceEPR>
    <wsa:Address>http://test.ibm.com:8080/websphere-management/services/datasource</wsa:Address>
    <wsa:ReferenceParameters>
     <was-wsdm:WAS_Resource_ManagedNodeID xmlns:was-wsdm="http://ibm.com/2006/v1.3"/>
     <was-wsdm:WAS_Resource_Type xmlns:was-wsdm="http://ibm.com/2006/v1.3">datasource</was-wsdm:WAS_Resource_Type>
     <was-wsdm:WAS_Resource_MbeanIdentifier xmlns:was-wsdm="http://ibm.com/2006/v1.3">WebSphere:name=Default Datasource,process=
server1,platform=dynamicproxy,node=Node01,JDBCProvider=Derby JDBC Provider,diagnosticProvider=true,j2eeType=JDBCDataSource,J2EEServer=
server1,Server=server1,version=7.0.0.0,type=DataSource,mbeanIdentifier=cells/Cell01/nodes/Node01/servers/server1/resources.xml#DataSource_1,
JDBCResource=Derby JDBC Provider,cell=Cell01,spec=1.0</was-wsdm:WAS_Resource_MbeanIdentifier>
     <was-wsdm:WAS_Resource_MRID xmlns:was-wsdm="http://ibm.com/2006/v1.3">datasource:WebSphere:name=Default Datasource,process=
server1,platform=dynamicproxy,node=Node01,JDBCProvider=Derby JDBC Provider,diagnosticProvider=true,j2eeType=JDBCDataSource,J2EEServer=
server1,Server=server1,version=7.0.0.0,type=DataSource,mbeanIdentifier=cells/Cell01/nodes/Node01/servers/server1/resources.xml#DataSource_1,
JDBCResource=Derby JDBC Provider,cell=Cell01,spec=1.0</was-wsdm:WAS_Resource_MRID>
     </wsa:ReferenceParameters>
     </wsrf-sg:MemberServiceEPR>
     <wsrf-sg:Content/>
     </wsrf-sg:Entry>
     </wsrf-rp:GetResourcePropertyResponse>
</env:Body>
</env:Envelope>
```

Request for concurrentRequests performance metrics from a servlet resource

This example shows how to request concurrentRequests metrics from a servlet resource. The resource that is of interest here is specified as reference parameters in the SOAP header. These parameters appear in bold font.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    <soap:Header>
        <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">http://test.ibm.com:8080/websphere-management/services/servlet</wsa:To>
        <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/
GetResourcePropertyRequest</wsa:Action>
        <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">uuid:2737b4e1-31af-5b0d-cb6e-17e2c812ffbb</wsa:MessageID>
        <wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
        <wsa:Address>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:Address>
        </wsa:From>
        <wsa-wsdm:WAS_Resource_ManagedNodeID xmlns:wsa="http://www.w3.org/2005/08/addressing" wsa:IsReferenceParameter=
"true" xmlns:was-wsdm="http://ibm.com/2006/v1.3">
<was-wsdm:WAS_Resource_Type xmlns:wsa="http://www.w3.org/2005/08/addressing" wsa:IsReferenceParameter="true" xmlns:was-wsdm=
"http://ibm.com/2006/v1.3">
        servlet</was-wsdm:WAS_Resource_Type>

<was-wsdm:WAS_Resource_MbeanIdentifier xmlns:wsa="http://www.w3.org/2005/08/addressing" wsa:IsReferenceParameter="true"
 xmlns:was-wsdm="http://ibm.com/2006/v1.3">WebSphere:WebModule=WSDMUtil.war,name=ejb,process=server1,Application=
wsdm-util,platform=dynamicproxy,J2EEApplication=wsdm-util,node=Node01,J2EEName=wsdm-util#WSDMUtil.war#ejb,j2eeType=
Servlet,J2EEServer=server1,Server=server1,version=7.0.0.0,type=Servlet,mbeanIdentifier=wsdm-util#WSDMUtil.war#ejb,cell=
Cell01,spec=1.0</was-wsdm:WAS_Resource_MbeanIdentifier>

<was-wsdm:WAS_Resource_MRID xmlns:wsa="http://www.w3.org/2005/08/addressing" wsa:IsReferenceParameter="true" xmlns:was-wsdm=
"http://ibm.com/2006/v1.3">servlet:WebSphere:WebModule=WSDMUtil.war,name=ejb,process=server1,Application=wsdm-util,platform=
dynamicproxy,J2EEApplication=wsdm-util,node=Node01,J2EEName=wsdm-util#WSDMUtil.war#ejb,j2eeType=Servlet,J2EEServer=server1,Server=
server1,version=7.0.0.0,type=Servlet,mbeanIdentifier=wsdm-util#WSDMUtil.war#ejb,cell=Cell01,spec=1.0</was-wsdm:WAS_Resource_MRID>

    <soap:Body>
<wsrf-rp:GetResourceProperty xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2" xmlns:servlet=
"http://www.ibm.com/xmlns/prod/websphere/management/j2ee/servlet">servlet:ConcurrentRequests</wsrf-rp:GetResourceProperty>
    </soap:Body>
  </soap:Envelope>
```

Number of concurrentRequests response

This example is the server response. The exact resource is identified in SOAP header and the metric is available in SOAP body.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
xmlns:soapenc="http://www.w3.org/2003/05/soap-encoding"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsd="http://www.w3.org/20001/XMLSchema-instance">
<env:Header>
<wsa:To>http://www.w3.org/2005/08/addressing/role/anonymous</wsa:To>
<wsa:Action>http://docs.oasis-open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyResponse</wsa:Action>
<wsa:MessageID>uuid:91400c29-8c50-9f62-7df2-1bb0212083f1</wsa:MessageID>
```

```
<wsa:RelatesTo RelationshipType="wsa:Reply">uuid:2737b4e1-31af-5b0d-cb6e-17e2c812ffbb</wsa:RelatesTo>
<wsa:From>
<wsa:Address>http://test.ibm.com:8080/websphere-management/services/servlet</wsa:Address>
<wsa:ReferenceParameters>
<was-wsdm:WAS_Resource_ManagedNodID xmlns:was-wsdm="http://ibm.com/2006/v1.3" IsReferenceParameter="true"
<was-wsdm:WAS_Resource_Type xmlns:was-wsdm="http://ibm.com/2006/v1.3" IsReferenceParameter="true">servlet</was-wsdm:WAS_Resource_Type>
<was-wsdm:WAS_Resource_MbeanIdentifier xmlns:was-wsdm="http://ibm.com/2006/v1.3" IsReferenceParameter="true">
WebSphere:WebModule=WSDMUtil.war,name=ejb,process=server1,Application=wsdm-util,platform=dynamicproxy,J2EEApplication=wsdm-util,node=
Node01,J2EEName=wsdm-util#WSDMUtil.war#ejb,j2eeType=Servlet,J2EEServer=server1,Server=server1,version=7.0.0.0,type=
Servlet,mbeanIdentifier=wsdm-util#WSDMUtil.war#ejb,cell=Cell01,spec=1.0</was-wsdm:WAS_Resource_MbeanIdentifier>
<was-wsdm:WAS_Resource_MRID xmlns:was-wsdm="http://ibm.com/2006/v1.3" IsReferenceParameter="true">servlet:WebSphere:WebModule=
WSDMUtil.war,name=ejb,process=server1,Application=wsdm-util,platform=dynamicproxy,J2EEApplication=wsdm-util,node=Node01,J2EEName=
wsdm-util#WSDMUtil.war#ejb,j2eeType=Servlet,J2EEServer=server1,Server=server1,version=7.0.0.0,type=Servlet,mbeanIdentifier=
wsdm-util#WSDMUtil.war#ejb,cell=Cell01,spec=1.0</was-wsdm:WAS_Resource_MRID>
</wsa:ReferenceParameters>
</wsa:From>
<wsa:RelatesTo>uuid:2737b4e1-31af-5b0d-cb6e-17e2c812ffbb</wsa:RelatesTo>
</env:Header>
<env:Body>
<wsrf-rp:GetResourcePropertyResponse xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
<servlet:ConcurrentRequests
xmlns:servlet="http://www.ibm.com/xmlns/prod/websphere/management/j2ee/servlet>11</servlet:ConcurrentRequests>
</wsrf-rp:GetResourcePropertyResponse>
</env:Body>
</env:Envelope>
```

Web Services Distributed Management

Web Services Distributed Management (WSDM) is an OASIS approved standard that supports managing resources through a standardized Web service interface. Your environment, such as WebSphere Application Server host or an operating system host that has an exposed resource as a Web service within a single interface is used to manage and control resources. WSDM is a distributed management model, but it does not replace any existing WebSphere Application Server administration models. WSDM provides a new way to expose the internal product administration functions for a Web service interface.

The existing administration interfaces, such as managed bean (MBean), wsadmin, and Java Application Programming Interface (API), are more language and platform specific. WSDM provides a common, flexible infrastructure to manage the product resources by leveraging the Web services protocols.

WSDM defines two specifications: Management Using Web Services (MUWS) and Management of Web Services (MOWS). MUWS defines how resources interact with the resources managed through a set of accessible Web services interfaces. MOWS extends the MUWS concepts to define how a Web service resource, itself, is managed. See Specifications and API documentation for MOWS and MUWS specifications. In addition to the manageability capabilities defined in the MUWS specifications, WebSphere Application Server WSDM also defines manageability capabilities unique to the product environment.

There is a general pattern that managed resources use to expose their manageability services through WSDM compliant Web services interfaces. First, you must create a model of the managed resource. Typically the model of the resource is created using a modeling tool such as the Test and Performance Tools Platform (TPTP), an eclipse plug-in tool; however, a simple text document is sufficient. Use the modeling tool to develop the model of WebSphere Application Server managed resources. The following graphic illustrates the process.

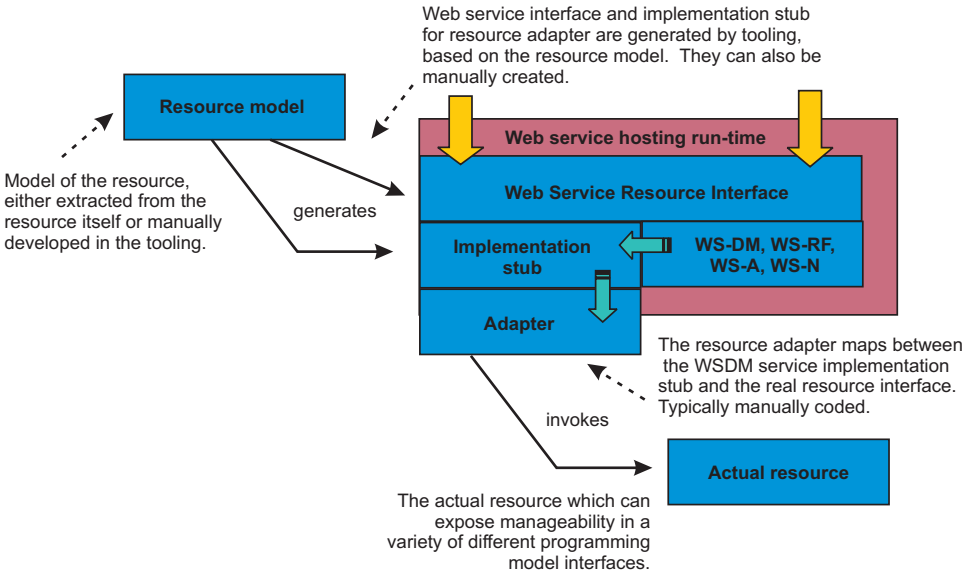


Figure 9. Generic WSDM Concept

Code artifacts are generated from the resource model. Generated artifacts for each resource model include:

- A Web Services Description Language (WSDL) document that describes the Web service interface for the management functions for that resource
- An implementation stub for the service implementation classes for that Web service
- A client proxy for the service that is used in a program that needs to invoke the management functions of that resource
- A unit test code for invoking test cases that exercises the functions of that service
- Additional XML documents and schema that describe the properties, operations, and notifications associated with the managed resource

The code generated from the resource model is essentially an empty shell of the management Web service for the modeled resource. The next step in the process is to enter code that acts as an adapter between the implementation stub for the service and the real resource management functions. In the case of the WSDM support implementation, this adapter code contain calls to theWebSphere Application Server AdminService APIs that expose normal product management functions. You must install the completed service implementation in a hosting Web service environment. To install your WSDM application, see Deploying and administering enterprise applications and follow the steps for installing enterprise application files on an application server.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

Web Services Distributed Management resource management

Web Services Distributed Management (WSDM) is an OASIS approved standard that supports the management of resources through a standardized Web service interface. WSDM delivers Web services based interfaces to manage application server resources using a manageability endpoint.

The manageability endpoint contains manageability capabilities for the resources. A manageability capability uniquely identifies and associates with a set of properties, operations, and events. A resource that supports one or more manageability capabilities becomes a manageable resource. For example, a

manageable resource is a server or an application resource that supports a capability, which includes stop, start, and remove operations. To leverage the functions that a manageable resource provides, a manageability consumer is used. Manageability consumer queries and discovers the available manageable resources through the Web services endpoints. After the service is discovered, the manageability consumer exchanges messages to gather property information, invoke operations or receive notifications. The following graphics illustrates the relationships between the manageability consumer and manageable resources linked by the Web service endpoints.

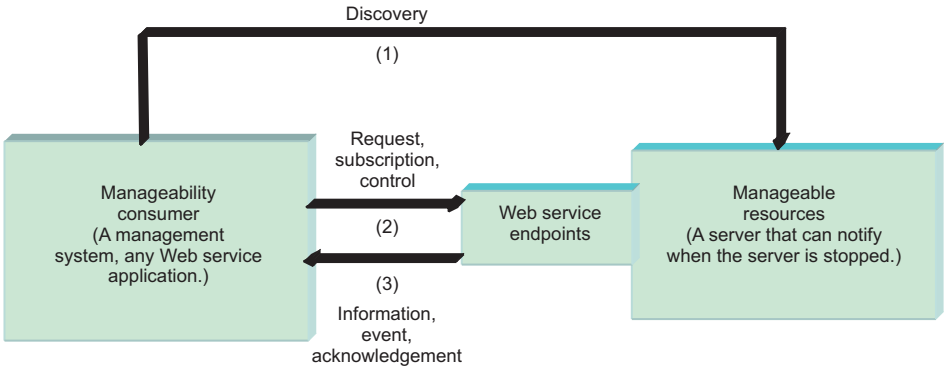


Figure 10. Relationship between the different parts of WSDM

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

WSDM manageability capabilities for WebSphere Application Server resource types

A resource that supports one or more manageability capabilities is a manageable resource. Each resource type that is exposed within the product supports a number of manageability capabilities.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

A manageable resource is a server or an application that supports a capability which includes stop, start, and remove operations. A manageability capability includes some properties, operations, and notifications. You can obtain and view performance data about the managed resources when you enable Performance Monitoring Infrastructure (PMI) in your server environment.

Resource types

WSDM manageable resources, in general, are an aggregation of manageability capabilities. There are manageability capabilities that are globally applicable to many resource types. State management fits into this category. There are manageability capabilities that are unique to a single-managed resource, for example the Java virtual machine (JVM) manageability capability only applies to JVM-managed resources.

The autonomic manager (AC), which can be any client with management capability interact with the resources. Before the AC can interact with the resources, the AC needs to query what resources are available in the application server via the service group. The service group is an aggregation of WS-Resources within the same domain. The WebSphere Application Server WSDM service group contains all of the resources. Each resource becomes a member in the service group. The AC can get a

particular resources endpoint reference (EPR) from the service group based on the resource type or the reference parameters. After the EPR is obtained, the AC can send the request to the resource. The service group can be accessed using the following endpoint address: http://<hostname>:<port>/websphere-management/services/service-group.

After the AC gets the resources EPR list from the service group, the AC can send requests to the resource provider. Each resource endpoint is listed below. The associated Web Service Definition Language (WSDL) can be obtained by attaching `?wsdl` to the end of the endpoint address.

Resource type	Resource endpoint address
WebSphere Application Server profile, also called runtime configuration instance or WebSphere Application Server domain	http://<hostname>:<port>/websphere-management/services/webspheredomain
WebSphere Application Server	http://<hostname>:<port>/websphere-management/services/applicationserver
WebSphere Application Server cluster	http://<hostname>:<port>/websphere-management/services/websphere cluster
Java virtual machine	http://<hostname>:<port>/websphere-management/services/jvm
Application	http://<hostname>:<port>/websphere-management/services/application
WebSphere Application Server deployed object	http://<hostname>:<port>/websphere-management/services/deployedobject
Servlet	http://<hostname>:<port>/websphere-management/services/servlet
Enterprise JavaBeans	http://<hostname>:<port>/websphere-management/services/ejb
Web services	http://<hostname>:<port>/websphere-management/services/webservices
JAX-WS Web services	http://<hostname>:<port>/websphere-management/services/jaxwswebservices
JAX-RPC Web services	http://<hostname>:<port>/websphere-management/services/jaxrpcwebservices
Data source	http://<hostname>:<port>/websphere-management/services/datasource

Manageability capabilities of the resource types

Each resource type that is exposed in the product supports a number of manageability capabilities. These resources are defined by the WSDM specification, AC touchpoint, and the product's built-in management. A touchpoint is a combination of port types and operations defined in WSDL that exposes the manageability interface for a managed resource in a way that complies with different specifications for Web services. Each manageability capability includes a number of properties, operations, and notifications.

The following table lists the manageability capabilities that each resource aggregates. For information about an Application Programming Interface (API) or a specification that is listed with a manageability capability, see Specifications and API documentation.

Resource types and manageability capabilities

Resource type	Manageability capabilities	Specification
WebSphere Application Server domain	• J2EEDomain • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType • Configuration • ApplicationManagement • ConfigChangeNotifier • NotificationProducer	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • MUWS – WSDM • WebSphere Application Server unique • WebSphere Application Server unique • WSBN - WS-N
WebSphere Application Server	• J2EEServer • J2EEManagedObject • Identity • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • NotificationProducer • ApplicationServer • StateManageable	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WSBN – WS-N • WebSphere Application Server unique • WebSphere Application Server unique
WebSphere Application Server cluster	• Identity • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • ClusterManagement	• MUWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique
Java virtual machine	• JVM • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint

Resource type	Manageability capabilities	Specification
Application	• J2EEApplication • J2EEDeployedObject • J2EEManagedObject • Identity • State • Metrics • ManageabilityCharacteristics • Description • ResourceType • Application • StateManageable	• JSR 77 – J2EE • JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS - WSDM • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • WebSphere Application Server unique
Servlet	• Servlet • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint
Enterprise JavaBeans	• EJB • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType	• JSR 77 – J2EE • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint
Web service	• Metrics • J2EEManagedObject • Identity • State • ManageabilityCharacteristics • Description • ResourceType • WebService	• MOWS – WSDM • JSR 77 – J2EE • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique

Resource type	Manageability capabilities	Specification
JAXWS Web services	• J2EEManagedObject • Identification • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • WebService • Manageability references • OperationalStatus • Operational state • Operation operational status • Request processing state • Identity	• JSR 77 – J2EE • MOWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MUWS – WSDM
JAXRPC Web services	• Metrics • J2EEManagedObject • Identification • Metrics • State • ManageabilityCharacteristics • Description • ResourceType • WebService • Manageability references • OperationalStatus • Operational state • Operation operational status • Request processing state • Identity	• MOWS – WSDM • JSR 77 – J2EE • MOWS – WSDM • MUWS – WSDM • MUWS – WSDM • MUWS • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MOWS – WSDM • MUWS – WSDM
Data source	• JDBCDatasource • J2EEResource • J2EEManagedObject • Identity • Metrics • ManageabilityCharacteristics • Description • ResourceType • DataSource	• JSR 77 – J2EE • JSR 77 – J2EE • JSR 77 – J2EE • MUWS - WSDM • MUWS - WSDM • MUWS – WSDM • MUWS – WSDM • AC touchpoint • WebSphere Application Server unique

Specific application server manageability capabilities

The following table lists the attributes and operations for the product's manageability capabilities.

Manageability Capabilities	Attributes	Operations
J2EEDomain	None	• String getAttribute(String, String) • String[] queryNames(String queryString)
J2EEManagedObject	• objectName • stateMangeable • eventProvider • statisticsProvider	None
ConfigChangeNotifier	None	None (however, it has notification of ConfigChange)
ApplicationManagement	None	• String installApplication(String, String, HashMap) • String uninstallApplication(String) • String updateApplication(String, String, HashMap) • String, HashMap • EndpointReference listApplications(String applicationName)
J2EEServer	• serverVendor • serverVersion • DepolyedObjects • javaVMs	None
StateManageable	• state • startTime	• stop() • start() • startRecursive()
ApplicationServer	• name • versionsForAllEFixes • versionsForAllExtensions • VersionsForAllPTFs • shortName • threadMonitorInterval • threadMonitorthreshold • threadMonitorAdjustmentThreshold • ProcessId • cellName • nodeName • processType • platformName • platformVersion	• stopImmediate() • restart() • String getproductVersion(String)

Manageability Capabilities	Attributes	Operations
ClusterManagement	• clusterName • preferLocal • wlcId • state • backupName • backupBootstrapHost • backupBootstrapPort	• start() • stop() • stopImmediate() • rippleStarT() • exportRouteTable() • dumpClusterInfo() • boolean getAvailable(String, String) • boolean setAvailable(String, String) • boolean setUnavailable(String, String)
Java virtual machine	• javaVersion • javaVendor • node • stats • freeMemory • usedMemory • heapSize • upTime • GCCount • GCTime • GCInternalTime • waitsForLockCount • waitForLockTime • objectAllocatedCount • objectMovedCount • objectFreedCount • threadStartedCount • threadEndedCount	None
J2EEDeployedObject	• deploymentDescriptor • server	None
J2EE Application	module	None
Application	implementationVersion	None
Servlet	• concurrentRequest • responseTime • numErrors • totalRequests	None
EJB	• createCount • loadCount • storeCount • readyCount • liveCount • pooledCount • waitTime	None

Manageability Capabilities	Attributes	Operations
WebService	• payloadSize • replyPayloadSize • requestPayloadSize • requestResponseTime • replyResponseTime • responseTime • processRequestCount • dispatchedRequestCount • receivedRequestCount • loadedWebServiceCount	None
DataSource	• jdbcDriver • connectionFactoryType • dataSourceName • dataStoreHelperClass • loginTimeout • statementCacheSize • jtaEnabled • name • jndiName • testConnection • testConnectionInterval • stuckTimerTime • stuckTime • stuckThreshold • surgeThrehold • surgeCreationInterval • connectionTimeout • maxConnections • minConnctions • purgePolicy • reapTime • unusedTimeout • agedTimeout • freePoolDistributionTableSize • freePoolPartions • sharedPoolPartitions	• String showPoolContents() • void purgePoolContents() • void pause() • void resume() • String getStatus()

Web Services Distributed Management support in the application server

The Web Services Distributed Management (WSDM) support for a Web service in WebSphere Application Server runs within an application server that has exposed management functions.

In the application server implementation of WSDM, a WSDM application is packaged as a Java Platform, Enterprise Edition (Java EE) Enterprise archive (EAR) file. The EAR file is deployed as an application server system application.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

WSDM support for the product consists of two parts:

- WSDM runtime environment and support
- WSDM resource model and service implementation

WSDM runtime environment and support

The WSDM runtime environment provides fundamental capabilities for the manageable resources. The WSDM runtime environment interacts with the underlying Web services platform and the WSDM resources to service the requests and responses. There are multiple specifications that the WSDM runtime environment uses in order to provide the WSDM functions, namely `WS-Addressing`, `WS-ResourceFramework`, and `WS-Notification`. For each request, the WSDM runtime environment routes the request to the appropriate resource service implementation based on the endpoint reference, (EPR). The EPR is defined by the `WS-Addressing` specification. Each EPR contains target address, runtime specific data and reference properties to uniquely identify an instance of a WSDM resource. After the resource service implementation returns a response, the WSDM runtime environment wraps the response into an appropriate SOAP message format specified in the Management Using Web Services (MUWS) specification and returns the response back to the requester. The application server leverages Apache Muse 2.0 to provide the runtime support for WSDM. The Apache MUSE 2.0 provides both the development tool and the WSDM runtime environment.

WSDM resource model and service implementation

The WSDM resource model for the application server identifies the elements of the product that are managed resources and further defines the specific properties, operations, and notifications that are managed resources support. The resource model defines the interfaces to interact with the resources and administrative functions in the product. The resource model includes appropriate capabilities defined in the two WSDM specifications, Management Using Web Services (MUWS) and Management of Web Services (MOWS). What this means is that the implementation is a mapping of the WSDM specification interfaces onto the product administration and programming interfaces. The implementation does not introduce new functions into the product, but rather, an alternative interface for accessing existing administration and programming functions in the product. In addition, the resource model defines specific capabilities to provide additional manageability functions. Each of the capabilities defines a set of properties, operations, and events for managed resources in an autonomically managed system. Each resource is associated with a Web Services Description Language (WSDL) file that contains the definition of its manageability capabilities.

Security

The implementation is attached to the WSSecurity default policy set and runs the administrative operations from the client user identity. This user identity must have privileges to perform any administrative action. It is the role of the autonomic computing (AC) manager that makes requests for the WSDM implementation to ensure that the user of that manager has appropriate authorization to perform administration and any other functions exposed by the AC manager.

The benefit of WSDM support in the application server is that the product can participate in multiple product management solutions in a standard way. By exposing the product management functions through a standard Web services interoperable interface, you can combine the application server with large management systems based on the WSDM specification.

Web Services Distributed Management in a standalone application server instance

In a standalone application server environment, there is one Web Services Distributed Management (WSDM) application deployed for each application server instance.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

The WSDM application acts as an administrative client to the management code running inside the single Java virtual machine for that instance. Figure 1 illustrates an Autonomic Computing Manager (ACM) interacting with two application server instances, each with its own WebSphere Application Server WSDM application exposing the manageability for that individual instance.

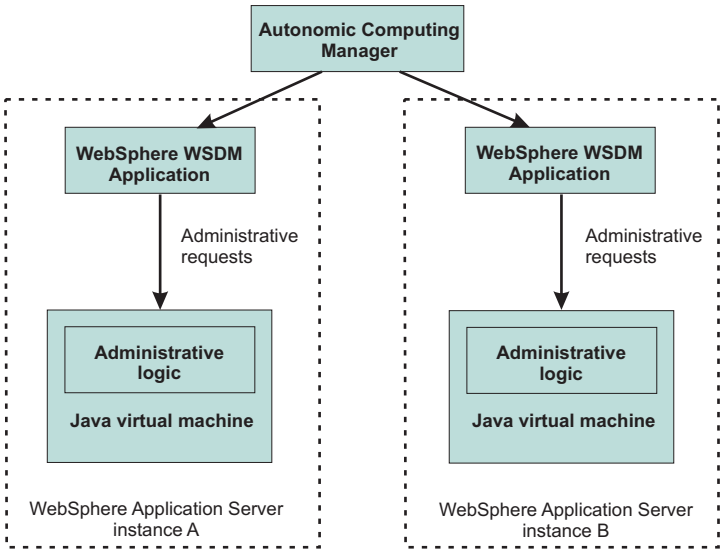


Figure 11. WSDM application in a stand-alone server instance

Web Services Distributed Management in a Network Deployment cell

You can use Web Services Distributed Management (WSDM) to manage application server instances within a Network Deployment cell. The administrative support and visibility for WSDM in a cell is obtained through interaction with each WSDM application deployed on the application server.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

In the multinode Network Deployment environment, the management code runs across a distributed network of Java virtual machines with a central access point as the deployment manager process for the entire network or cell. Several different application server Java virtual machines might be managed within a cell. You can manage an application server Java virtual machines within a cell through the WSDM application installed on deployment manager. The WSDM application acts as an administrative client to the managed application server. Figure 1 illustrates this environment with an Autonomic Computing Manager interacting with the single application server implementation of WSDM to expose the manageability of that application server. You can build a federated deployment manager cell from individual application server instances by running the addNode utility program to add the application server instances to the centrally

managed cell. After a node is added to the cell, the manager can still manage each application server within the cell through the installed WSDM application on the deployment manager.

Figure 12. WSDM application in a Network Deployment cell

Web Services Distributed Management in an administrative agent environment

You can use Web Services Distributed Management (WSDM) to manage application server profiles in an administrative agent (AdminAgent) environment.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

You can manage multiple base application servers within profiles using the AdminAgent. You can also use the Web Services Distributed Management (WSDM) to manage each profile within the AdminAgent. WSDM is deployed inside the AdminAgent as a system application. Each profile is represented as a domain resource within the product. You can get and manage the resources within a profile through the product's domain resource. Figure 1 illustrates the AdminAgent topology.

Figure 13. WSDM application in an AdminAgent environment

Notifications from the application server Web Services Distributed Management resources

Use this topic to learn about application server Web Services Distributed Management (WSDM) resources and their life cycle events.

Note: WSDM is a system application and it is disabled by default when the product is installed. You must first enable WSDM before you can use it to manage the product resources. Use scripting to enable WSDM.

There are different life cycle events that can occur for any resource. The notifications associated with these lifecycle events are resource definition events and resource state events. Resource definition events are:

- Created
- Deleted
- Changed

Resource state events are:

- Started
- Stopped
- Suspended

The following resources are discussed in detail. You can review WSDM manageability capabilities for application server resource types for information about each resource endpoint address and manageability capabilities.

Application server

The application server instances support definitional notifications. Whenever a resource definition event occurs, the configuration of an application server is created, modified, or deleted and a notification is generated by the WebSphere Application Server domain. The notification includes the configuration documents that have been changed.

Application server installation

The product installation is performed and managed by the underlying operating system. There is no runtime entity that represents the product installation. There are no life cycle notifications currently planned for events related to the product installation. There are no state event notifications associated with the lifecycle of the overall product installation.

Application server domain: profile or runtime configuration instance

There is an administrative agent process that is created as part of a WebSphere Application Server profile. This administrative agent process, once created, becomes available to emit life cycle notifications for the profile. Since any configuration modification might be considered a change to the profile, there is no generic *profile changed* notification. Instead, there are specific notifications for some configuration changes. In addition, there is no state change notification for profiles because a profile does not actually run, it simply exists or does not exist.

Hosted applications

Applications that are installed into the product support both definitional and operational notifications. Whenever an application is installed, a notification is produced to indicate that an instance of that application managed resource has been created. A notification is generated each time the application is started, stopped, or updated. When an application is uninstalled, the *resource destroyed* notification is produced.

Application server deployed object

Individual deployed modules have independent life cycles. There are create, modify, delete, start, and stop notifications for individual deployed objects in the product.

Web services

Even though Web services are not robust applications, there is a need to understand the life cycle for these essential deployed objects. Notifications are produced in accordance to the MOWS specification such as when a Web service is installed, modified, started, stopped or uninstalled.

Managing policy sets using the administrative console

You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

Before you begin

Before creating policy sets, first identify the security and other requirements of the Web service.

Note: You can only use policy sets with JAX-WS applications that run on the Axis2 Web service engine. You cannot use policy sets for JAX-RPC applications.

About this task

You can use the administrative console to view and manage policy sets. From the administrative console, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**. The Application policy sets collection displays a listing of the custom (if you have created custom policy sets) and default policy sets. Use the Application or System policy sets collection page to create, copy, delete, export, and import policy sets.

The following policy sets are ready for you to use as is.

- LTPA WSSecurity Default
- Kerberos V5 HTTPS default
- SSL WSTransaction
- Username SecureConversation
- Username WSSecurity default
- WS-Addressing default
- WSHTTPS default
- WS-I RSP ND
- WS-ReliableMessaging persistent

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

- "Viewing policy sets using the administrative console" on page 805.

  This topic describes the process of viewing and evaluating policy sets.
- "Creating policy sets using the administrative console" on page 806.

  This topic describes two ways to create policy sets: creating new policy sets or copying and renaming policy set templates.
- "Modifying policy sets using the administrative console" on page 820.

  This topic describes how to edit custom policy sets you have created.
- "Importing policy sets using the administrative console" on page 816.

  This topic describes how to import policy sets from the default repository or from a selected location.
- "Exporting policy sets using the administrative console" on page 953.

  This topic describes how to export policy sets.
- "Deleting policy sets using the administrative console" on page 822.

  This topic describes how to delete custom policy sets. You can delete policy set templates and re-import them if needed.
- "Managing policies in a policy set using the administrative console" on page 902

  This topic describes how you can define policies with policy sets to secure messages.
- "Defining and managing policy set bindings" on page 824

  This topic describes configuring custom binding configurations.

Results

Using these tasks, you can determine how to create a new policy set and verify whether you can reuse an existing policy set. You can configure a policy set, and define policies for that policy set.

What to do next

Depending on how you are using policy sets, you might want to revisit some of the tasks listed in this topic to tweak the configuration for your policy set. You can also proceed to configure bindings for your policy set. See Defining binding information for policy sets.

Viewing policy sets using the administrative console

You can use the administrative console to view lists of policy sets. Policy sets can either be default policy sets that you cannot edit or custom policy sets that you have created and can edit. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

Before you begin

If you are creating a custom policy set by copying an existing default policy set, you might want to view the existing policy sets to choose a policy set with properties similar to the one you are planning to create.

About this task

You can view a list of policy sets to decide which policy sets can be reused or copied, modified and reused.

1. To view policy sets from the administrative console, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**.
2. Set the number of policy set rows that you want to view at a time using the Preferences settings. The default value is 20. The Editable column on the table in this view shows which policy sets you can edit. You can only edit the custom policy sets that you create. You cannot edit the provided default policy sets.
3. To view the details for any of the policy sets, click the name of the policy set in the Name column of the table. The Policy set settings page displays details about the selected policy set. If the policy set is a custom policy set you can edit the fields on the page. If the policy set is a provided default, then you can copy and reuse the policy set. You cannot edit the fields otherwise.

Results

After you have viewed the available policy sets and their settings, you can then decide if you want to create a new policy set, copy an existing policy set that you can rename, or use one of the existing policy sets that meets your needs.

Related tasks

"Creating policy sets using the administrative console"
You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Modifying policy sets using the administrative console" on page 820
You can use the administrative console to modify existing custom policy sets that you have created. If you have copied an existing default policy set or created a policy set yourself, you can always go back and make changes to them to make them better suit the changing needs of your business.

Related reference

"Application policy sets collection" on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Application policy set settings" on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Creating policy sets using the administrative console

You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

Before you begin

To create a new policy set, you can either specify the information to create a new policy set or you can copy and rename an existing policy set. Using either method, you need basic information about the policy set that you want to create, such as the name, description, policies to include, policy details, attachments, and binding configurations. If you are creating a policy set by copying an existing policy set, then you should also view the existing policy sets to choose one with properties that are most similar to the one you plan to create.

About this task

Whether you choose to create a new policy set or copy and rename an existing one, start from the Applications policy sets collection in the administrative console.

1. From the administrative console, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**.
2. If the policy set you are creating is:
   - a new policy set, then click **New**.
   - an existing policy set to be copied and renamed, click the **Select** box beside the name of the policy set to be copied in the**Name** column and click **Copy**.

   Using either method, this action opens the Policy set settings view to specify the required information about the policy set being created or copied.
3. Enter the name of the policy set that you want to create or copy in the **Name** field.
4. Enter a brief description of the policy set in the **Description** field. This is the description that displays in the Application policy sets or the System policy set collection, so it must be meaningful to you and other potential users of this policy set.

5. Optional: If you created a new policy set, you do not have any policies to edit until you add them to the policy set. The policy set is initially empty.

Results

You have provided the basic information to create a policy set.

Example

After you have looked at your Web services, you might decide that the WS-I RSP default policy set most closely meets your needs. You would go to the administrative console and click **Services > Policy sets > Application policy sets** to access the Application policy sets collection. Locate the WS-I RSP default in the Name column of the table and click the box beside it (in the Select column). Click **Copy**. This opens the Policy set settings window. You might want to name your policy set by your company or division so you could provide a name like ABC WS-I RSP in the Name field. Because you know others in your organization might access and use it, you've chosen a name that is meaningful to those people too. You want to be sure everyone knows exactly what this copy of the WS-I RSP policy is used for, so you add a description in the Description field describing it. Now you want to customize the policy set so you edit the policy information by clicking the name of a policy to edit it.

When you identify the requirements of your web service, you might decide that none of the default policy sets meet your needs closely enough to use them as a template so you might decide to create your own policy set. You would first create the policy set with the name you choose to give it. As if you were reusing an existing template, you would go to the administrative console and click **Services > Policy sets > Application policy sets** and click **New**. The Policy set settings window opens but note that the Policy set name field is blank and there are not yet any associated policies in the table. Enter the name and add any policies necessary.

When you add policies to a policy set, the policies are set to their default values. You can then edit the policies to modify any attribute values that need to be changed and save the settings.

What to do next

If you are creating a new policy set without copying an existing policy set, you need to specify the policy information. If you are copying an existing policy set, you can either accept the default policies associated with the policy set or you can change the policies.

Related concepts

Web Services Addressing support
The Web Services Addressing (WS-Addressing) support in this product provides the environment for Web services that use the Worldwide Web Consortium (W3C) WS-Addressing specifications. This family of specifications provide transport-neutral mechanisms to address Web services and to facilitate end-to-end addressing.

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

"Web services policies" on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

Related tasks

"Modifying policy sets using the administrative console" on page 820
You can use the administrative console to modify existing custom policy sets that you have created. If you have copied an existing default policy set or created a policy set yourself, you can always go back and make changes to them to make them better suit the changing needs of your business.

"Deleting policy sets using the administrative console" on page 822
You can use the administrative console to delete the default policy sets or the application specific policy sets that you have created.

"Managing policies in a policy set using the administrative console" on page 902
When working with policy sets in the administrative console, you can customize the included policies to ensure message security. You can enable, disable, customize, add, or delete policies from a policy set. With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

"Creating policy sets using the wsadmin tool" on page 498
Create policy sets to centrally manage policies that are customized for your Web services. Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to create new policy sets, copy existing policy sets, or import a policy set configuration. You can also query for an existing policy set and respective attributes.

"Creating policy set attachments using the wsadmin tool" on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

"Removing policy set attachments using the wsadmin tool" on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

"Managing policy set attachments using the wsadmin tool" on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Related reference

"Application policy sets collection" on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Application policy set settings" on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

WS-I RSP default policy sets

The Reliable Asynchronous Message Profile (WS-I RSP) default policy sets are based on the Reliable Asynchronous Message Profile specification. The WS-I RSP default policy sets include the WS-I RSP default policy set, the Lightweight Third-Party Authentication (LTPA) WS-I RSP default policy set and the Username WS-I RSP default policy set. You can use these policy sets to simplify your Web services configuration.

The WS-I RSP default policy sets are composed of a set of policies to provide reliable and secure Web services. The WS-I RSP default policy sets use the WS-Addressing, WS-ReliableMessaging, and WS-Security specifications. Use the WS-I RSP default policy set, the LTPA WS-I RSP default policy set, or the Username WS-Security WS-I RSP default policy set as provided with the application server. To customize the policy sets, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

The WS-I RSP default policy sets include the following policies:

WS-Addressing policy
: You can use the WS-Addressing policy to enable the addressing capability of the WS-Addressing specification.

WS-ReliableMessaging policy
: You can use the WS-ReliableMessaging policy to specify the quality of service for reliable delivery.

WS-Security policy
: The WS-Security policy in the WS-I RSP default policy set provides the following security:
  - Message integrity through digital signature that includes signing the body, time stamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications.
  - Confidentiality through encryption that includes encrypting the body, signature elements, using the WS-SecureConversation and WS-Security specifications.
  - Traditional RSA cryptography is used to secure a request to a Trust Server to obtain a Secure Context Token (SCT). Thereafter, the conversation is secured using symmetric keys derived from the SCT.

The application server provides additional policy sets that you can use or customize. To use the following default policy sets, you must import the policy sets from the default repository. Read about importing policy sets using the administrative console for more information.

The following WS-I RSP default policy sets exist:

WS-I RSP default
: This policy set provides:
  - Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging.
  - Message integrity through digital signature that includes signing the body, time stamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications.
  - Confidentiality through encryption that includes encrypting the body, signature elements, using the WS-SecureConversation and WS-Security specifications.

LTPA WS-I RSP default
: This policy set provides the WS-I RSP default policy set and adds a Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

Username WS-I RSP default

This policy set provides the WS-I RSP default policy set and adds a username token included in the request message to authenticate the client to the service. The username token is encrypted in the request.

Related concepts

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Related tasks

"Importing policy sets using the administrative console" on page 816
You can import predefined policy sets or import policy sets from a selected location using the administrative console.

SecureConversation default policy sets

The SecureConversation default policy sets are based on the Web Services Secure Conversation Language (SecureConversation) standard that establishes a secure context, based on shared keys for the client and server to use for a series of messages. This standard provides a framework to define how to secure the message exchange across organizations. The SecureConversation default policy sets include the SecureConversation policy set, the Lightweight Third-Party Authentication (LTPA) SecureConversation policy set, and the Username SecureConversation policy set.

The SecureConversation default policy sets are based on the WS-SecureConversation, the WS-Security, and the WS-Addressing specifications. Use the SecureConversation policy set, the LTPA SecureConversation policy set, or the Username SecureConversation policy set as provided with the application server. To customize the policy sets, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

The WS-SecureConversation specification alone does not provide a complete security solution. The WS-SecureConversation is built on the WS-Security and WS-Trust specifications to provide secure communication across one or more messages. Specifically, this specification defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts or any shared secret.

WS-Security focuses on the message authentication model but not in a security context. The WS-SecureConversation specification defines mechanisms for establishing and sharing security contexts, and deriving keys from security contexts, to enable a secure conversation. By using the SOAP extensibility model, modular SOAP-based specifications are designed to be composed with each other to provide a rich messaging environment.

The following SecureConversation default policy sets exist:

SecureConversation

This policy set provides:

- Message integrity by digital signature that includes signing the body, timestamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications.
- Message confidentiality by encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications.

LTPA SecureConversation

This policy set provides the SecureConversation policy set and adds a Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

Username SecureConversation

This policy set provides the SecureConversation policy set and adds a username token included in the request message to authenticate the client to the service. The username token is encrypted in the request

Related concepts

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

"WS-I RSP default policy sets" on page 809
The Reliable Asynchronous Message Profile (WS-I RSP) default policy sets are based on the Reliable Asynchronous Message Profile specification. The WS-I RSP default policy sets include the WS-I RSP default policy set, the Lightweight Third-Party Authentication (LTPA) WS-I RSP default policy set and the Username WS-I RSP default policy set. You can use these policy sets to simplify your Web services configuration.

WS-ReliableMessaging default policy sets

The WS-ReliableMessaging default policy sets are pre-configured to provide reliable message exchange between Web services. Two of these policy sets (WS-I RSP and WS-I RSP ND) are immediately available, and the rest are readily available for import from a default repository.

With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services reliable without writing custom code. You can use the provided non-editable default policy sets without change, or you can create customized copies of them.

All the default policy sets that include the WS-ReliableMessaging policy also include the WS-Addressing policy. The WS-ReliableMessaging policy provides the ability to deliver a message reliably to its intended receiver. The WS-Addressing policy provides a transport-neutral way to uniformly address Web services and messages, and WS-ReliableMessaging uses WS-Addressing to provide asynchronous request and reply capabilities.

Note: WS-ReliableMessaging Version 1.1 messaging requires WS-Addressing to be mandatory. If you use a policy set that includes WS-ReliableMessaging and WS-Addressing policies, and the WS-Addressing policy is configured as optional, then WebSphere Application Server overrides the WS-Addressing setting and automatically enables WS-Addressing.

The following default policy sets that include the WS-ReliableMessaging policy are immediately available, as described in "Viewing policy sets using the administrative console" on page 805:

WS-I RSP

This policy set enables WS-ReliableMessaging Version 1.1 and uses the minimum quality of service, *unmanaged non-persistent*. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent. Message integrity is provided by digitally signing the body, the time stamp, and the WS-Addressing headers. Message confidentiality is provided by encrypting the body and the signature. This policy set follows the WS-SecureConversation and WS-Security specifications.

WS-I RSP ND

This is the network deployment version of the WS-I RSP policy set. This policy set provides the WS-I RSP default policy set and adds a *managed non-persistent* quality of service. This in-memory quality of service option uses a messaging engine to manage the sequence state, and messages are written to disk if memory is low. This quality of service allows for the re-sending of messages that are lost in the network, and can also recover from server failure. However, state is discarded after a messaging engine restart so in this case you will lose messages.

The following additional default policy sets that include the WS-ReliableMessaging policy are readily available for import, as described in “Importing policy sets using the administrative console” on page 816:

LTPA WS-I RSP
This policy set provides the WS-I RSP default policy set and adds a Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

Username WS-I RSP
This policy set provides the WS-I RSP default policy set and adds a username token included in the request message to authenticate the client to the service. The username token is encrypted in the request.

WSReliableMessaging 1_0
This policy set enables both WS-ReliableMessaging Version 1.0 and WS-Addressing and uses the minimum quality of service, *unmanaged non-persistent*. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to “false”, so messages are not necessarily delivered in the order in which they were sent.

You can use this policy set with .NET-based Web services.

WSReliableMessaging default
This policy set enables both WS-ReliableMessaging Version 1.1 and WS-Addressing and uses the minimum quality of service, *unmanaged non-persistent*. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to “false”, so messages are not necessarily delivered in the order in which they were sent.

WSReliableMessaging persistent
This policy set enables both WS-ReliableMessaging and WS-Addressing and uses the maximum quality of service, *managed persistent*. This quality of service supports asynchronous Web service invocations and uses a service integration messaging engine and message store to manage the sequence state. Messages are processed within transactions, are persisted at the Web service requester server and at the Web service provider server, and are recoverable in the event of server failure. In-order delivery is set to “false”, so messages are not necessarily delivered in the order in which they were sent.

Because this policy set specifies managed persistent quality of service, you need to define bindings to the service integration bus and messaging engine that you want to use to manage the WS-ReliableMessaging state. For more information, see “Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console” on page 657 or using the wsadmin tool.

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

“WS-I RSP default policy sets” on page 809
The Reliable Asynchronous Message Profile (WS-I RSP) default policy sets are based on the Reliable Asynchronous Message Profile specification. The WS-I RSP default policy sets include the WS-I RSP default policy set, the Lightweight Third-Party Authentication (LTPA) WS-I RSP default policy set and the Username WS-I RSP default policy set. You can use these policy sets to simplify your Web services configuration.

Related tasks

“Adding assured delivery to Web services through WS-ReliableMessaging” on page 634
WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. With WS-ReliableMessaging, you can make your SOAP over HTTP-based Web services reliable without having to write custom code. You can get different qualities of service with WS-ReliableMessaging. These range from protecting against loss of messages across a network, through to protecting against a server becoming unavailable.

“Learning about WS-ReliableMessaging” on page 635
WS-ReliableMessaging is an interoperability standard for the reliable transmission of messages between two endpoints. Use these topics to learn more about WS-ReliableMessaging.

WSAddressing default policy set

The WSAddressing default policy set provides a transport-neutral way to uniformly address Web services and messages.

The WSAddressing default policy set is based on the WS-Addressing specification. The WS-Addressing standard uses endpoint references and message addressing properties to facilitate the addressing of Web services in a standard and interoperable way.

Use the WSAddressing default policy set as provided with the application server. To customize the policy set, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

To learn more about the WS-Addressing standard, read about Web Services Addressing support.

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Web Services Addressing support
The Web Services Addressing (WS-Addressing) support in this product provides the environment for Web services that use the Worldwide Web Consortium (W3C) WS-Addressing specifications. This family of specifications provide transport-neutral mechanisms to address Web services and to facilitate end-to-end addressing.

WSSecurity default policy sets

The WSSecurity default policy sets are based on the Web Services Security (WS-Security) 1.0 and Web Services Addressing (WS-Addressing) specifications. The WSSecurity default policy sets include the WSSecurity default policy set, the Lightweight Third-Party Authentication (LTPA) WSSecurity policy set, and the Username WSSecurity policy set. Use the WSSecurity default policy sets to build secure Web services.

The WSSecurity default policy sets use the WS-Security 1.0 specification enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. Providing quality of protection means to prevent the following potential threats to SOAP messages:

- The message being modified or read by antagonists
- An antagonist sending messages to a service that are formed correctly, but lack the appropriate security claims to be processed

The WSSecurity default policy sets provide message protection by using WS-Security to digitally sign the WS-Addressing headers, the time stamp and the body. This policy set also encrypts the signature and the body. RSA public key cryptography is used for the signature and for encryption operations.

The WS-Addressing specification defines XML 1.0 and XML Namespaces elements to identify Web services endpoints and to secure end-to-end endpoint identification in messages.

Use the WSSecurity default policy set, the LTPA WSSecurity policy set, or the Username WSSecurity policy set as provided with the application server. To customize the policy sets, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

The following WSSecurity default policy sets exist:

WSSecurity default
This policy set provides:

- Message integrity through digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications.
- Message confidentiality through encryption (using RSA public-key cryptography) to encrypt the body, and signature elements using WS-Security specifications.

LTPA WSSecurity default
This policy set provides the WSSecurity default policy set and adds aLightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

Username WSSecurity default
This policy set provides the WSSecurity default policy set and adds a username token included in the request message to authenticate the client to the service. The username token is encrypted in the request.

Related concepts

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

WSTransaction default policy sets

The WSTransaction default policy sets are based on the WS-Transaction specification and provide transactional integrity for Web services. The WSTransaction default policy sets include the WSTransaction policy set and the SSL WSTransaction policy set.

You can use the WSTransaction default policy sets to make your SOAP over HTTP-based Web services interoperable and coordinate atomic transactions or business activities without writing custom code. Use the WSTransaction policy set or the SSL WSTransaction policy set as provided with the application server. To customize the policy sets, you must first copy the policy set, and then configure custom policy settings and bindings to meet your needs.

The WSTransaction default policy sets are:

WSTransaction
Use this policy set to coordinate distributed transactional work atomically and interoperably using

the WS-AtomicTransaction specification. Also, use this policy set to coordinate loosely coupled business processes that are distributed across the heterogenous Web service environment, with the ability to compensate actions if a failure occurs in the business activity, using the WS-BusinessActivity specification.

SSL WSTransaction

Use this policy set to coordinate distributed transactional work atomically, interoperably and securely using the WS-AtomicTransaction specification and SSL Transport security. Also, use this policy set to coordinate loosely coupled business processes, with the ability to compensate actions if a failure occurs in the business activity, securely, using the WS-BusinessActivity specification and SSL Transport security.

Related concepts

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

"Web Services Atomic Transaction support in the application server" on page 689
The Web Services Atomic Transaction (WS-AT) support in the application server provides transactional quality of service to the Web services environment. Distributed Web services applications, and the resources they use, can take part in distributed global transactions.

"Web Services Business Activity support in the application server" on page 693
With Web Services Business Activity (WS-BA) support in the application server, Web services on different systems can coordinate activities that are more loosely coupled than atomic transactions. Such activities can be difficult or impossible to roll back atomically, and therefore require a compensation process if an error occurs.

WSHTTPS default policy set

The WSHTTPS default policy set provides SSL transport security for the HTTP protocol with Web services applications.

The WSHTTPS default policy set is provided with the application server and it contains the HTTP transport policy, the SSL transport policy and the WS-Addressing policy.

You can use the WSHTTPS default policy set as provided with the application server. You cannot edit the WSHTTPS default policy set. You can create a copy of the default policy set and then configure custom policy settings and bindings to meet your needs. Alternatively, you can create a new policy set and specify the policies for it.

Related concepts

"Web services policy sets" on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Copy of default policy set and bindings settings

Use this page to copy a policy set that you select from a list of available policy sets.

To view this administrative console page:

1. Click **Services > Policy sets > Application policy sets**.
2. Select the policy set that you want to copy, and click **Copy**.
3. Enter a name for the copy of the policy set in the **Name** field.
4. [Optional] Enter a description for the copy of the policy set in the **Description** field.

Name:

Specifies the name of the policy set. Use this field to enter a name for the copy of the policy set.

Description:

Specifies a description of the policy set that you want to copy.

Transfer attachments:

If the policy set that you want to copy is attached to one or more applications, services, or endpoints, then the check box option for **Attach this policy set in place of the original for all attached applications, services, endpoints, and operations**, is available. The default setting for this check box is cleared. You can perform the following actions:

1. Select the **Attach this policy set in place of the original for all attached applications, services, endpoints, and operations** check box to move the attachments to a new policy set. Selecting the check box detaches the original policy set and attaches the replacement policy set in its place.
2. Select **Copy bindings** if you want to copy the bindings of the policy set that is currently attached.
3. [Optional] Select **Restore default bindings** if you want to restore the default bindings.

Importing policy sets using the administrative console

You can import predefined policy sets or import policy sets from a selected location using the administrative console.

Before you begin

Before you begin this task, review Application policy sets collection.

About this task

The policy sets panel has an Import button to allow the import of policy sets. You can import a policy set from the default repository or from a selected location.

1. From the administrative console navigation, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**. The policy sets collection page displays a listing of the custom and default policy sets. Custom policy sets are displayed only if you have created them. If you have not created a custom policy set, then only the default policy sets are displayed.
2. From the Application policy sets panel, click **Import**.
3. Choose the option for importing the policy set. You can choose to import a policy set from the default repository or from a selected location. Selecting **Import from Default Repository** accesses the next panel that displays a set of predefined default policy sets that are available for you to import. Some of the predefined policy sets have already been imported for you from the default repository. You can import additional predefined policy sets that are available in the predefined policy set repository.

   Select **Import from Selected Location** to provide a path or browse your file system for a policy set file to import.

   - To import a policy set from the default repository, select **Import from Default Repository** and perform task step 4.
   - To import a policy set from a selected location, select **Import from Selected Location** and perform task step 7.
4. Select **Import from Default Repository** and select either the **Import** or the **Import a Copy** radio button.

   The behavior of **Import** policy set:

   - Available for selection of one or more policy sets.
   - Imports the default, not editable, policy set. You are returned to the Application policy sets panel after the import of the policy set.
   - The imported policy set can be deleted, but it cannot be modified.

   The behavior of **Import a Copy** for a policy set:

- Not available for multiple selection. If multiple policy sets are selected and the **Import a Copy** radio button is selected, an error message is displayed, indicating that only one policy set can be copied at a time.
- Prompts you for a name and description for the copy. You are returned to the Application policy sets panel after you import the policy set.
- The imported policy set can be modified or deleted.

5. If you selected **Import** from the previous step, select one or more policy sets to import in the **Name** column.
   a. Click **OK**. You are returned to the Application policy set panel.
   b. Click **Save**, to save your changes to the configuration.
6. If you selected **Import a Copy** from step 4, select the policy set to import a copy in the **Name** column. You can only select one policy set in the Name column to import its copy.
   a. Provide a different name for the copy.
   b. Click **OK**. You are returned to the Application policy set panel.
   c. Click **Save**, to save your changes to the configuration.
7. Select **Import from Selected Location**.
   a. Specify the full path or browse for the policy set file in your file system.
   b. Select either **Use current policy set name** or **Specify a different name to use for this policy set**.
   c. Click **OK**. You are returned to the Application policy set panel.
   d. Click **Save** to save your changes to the configuration.

Results

When you finish this task, you have been able to import a policy set using either the default repository or from a selected location.

Example

If you have a policy set, `XYZ_ps` and you want to import it from a selected location, perform the following steps:

1. From the Application policy set panel, click **Import**.
2. Select **Import from Selected Location**.
3. Specify the full path or browse to where the `XYZ_ps` policy set file is located in your file system.
4. Select either **Use the current policy set name**, `XYZ_ps`, or **Specify a different name for the policy set**.
5. Click **OK**.
6. Click **Save**, to save your changes to the configuration.

What to do next

You can modify or use the policy set that you just imported.

Related tasks

“Importing and exporting policy sets to client or server environments using scripting” on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

Import policy sets from default repository settings

Use this page to specify a policy set to import from a default repository. You can import predefined policy sets from the default repository.

To view this administrative console page, complete the following actions:

1. Click **Services > Policy sets > Application policy sets**.
2. Click **Import**.
3. Select **From Default Repository**.
4. You have the Import and Import a copy options. Select **Import a copy**.
5. In the **Name of copy** field, enter a name for the copy of the policy set that you select to import from the list of policy sets.

The behavior of **Import**:

- Available for selection of one or more policy sets.
- Imports the default, not editable, policy set. You are returned to the Application policy sets panel after the import of the policy set.
- The imported policy set can be deleted, but it cannot be modified.

The behavior of **Import a copy** policy set:

- Not available for multiple selection. If multiple policy sets are selected and the **Import a copy** radio button is selected, an error message is displayed, indicating that only one policy set can be copied at a time.
- Prompts you for a name for the copy. The description of the imported policy set is kept, but you can edit the description after you have imported the policy set. You are returned to the Application policy sets panel after the import of the policy set.
- The imported policy set can be modified or deleted.

Related tasks

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Importing and exporting policy sets to client or server environments using scripting” on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Import policy sets from a selected location settings”
Use this page to specify a policy set to import from a selected location.

Import a copy:

Use this field to enter a name for the copy of the policy set that you select to import from the list.

Before you can edit this field, you must select **Import a copy** option.

Buttons:

Click the appropriate button:

Button	Resulting Action
OK	Imports the specified policy set. You are returned to the Application policy set panel after the import.
Cancel	Returns you to the Application policy set panel without importing a policy set.

Import policy sets from a selected location settings

Use this page to specify a policy set to import from a selected location.

To view this administrative console page, click **Services > Policy sets > Application policy sets > Import policy set from selected location**.

On this page, specify a policy set to import from a selected location.

Related tasks

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Importing and exporting policy sets to client or server environments using scripting” on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Import policy sets from default repository settings” on page 818
Use this page to specify a policy set to import from a default repository. You can import predefined policy sets from the default repository.

Path and file name for the policy set file:

Specify a fully qualified path and file name to the policy set file or use the **Browse** to locate the file in your file system.

Select **Use current policy set name** if you are not changing the policy set name and click **OK**.

Select **Specify a different name for this policy set** if you are changing the policy set name. Provide a new name and click **OK**.

Button	Resulting action
OK	Imports the specified policy set file if valid or returns an error if an invalid file is provided. You are returned to the Application policy set panel after the import.
Cancel	Returns you to the Application policy set panel without importing a policy set.

Modifying policy sets using the administrative console

You can use the administrative console to modify existing custom policy sets that you have created. If you have copied an existing default policy set or created a policy set yourself, you can always go back and make changes to them to make them better suit the changing needs of your business.

Before you begin

You can create a copy of a policy set and modify the copy. You must first copy a default policy set before you can modify it. See copy of default policy set and bindings settings.

About this task

To modify a custom policy set, you can use the Policy set settings view to access configuration information for the policy set.

1. Create a policy set using the steps in “Creating policy sets using the administrative console” on page 806
2. From the administrative console, click **Services > Policy sets > Application policy sets** >*policy_set_name* or **Services > Policy sets > System policy sets** >*policy_set_name* where

policy_set_name is the custom policy set of interest that is displayed in the table. This opens the Policy set settings view. You must specify the required information about the policy set that you want to modify.

3. Change the description of the policy set. Edit the brief description of the policy set in the Description field. This is the description that displays in the Application policy sets collection so it must be meaningful to you and other potential users of this policy set.
4. Edit the policy information. You can include, exclude, add or delete policies from this policy set using the settings in the Policies table. To work more extensively with policies, see “Managing policies in a policy set using the administrative console” on page 902.
5. This step does not apply to system policy sets. Detach the policy set from an application or replace the policy set attached to the application with another policy set. Click the **Attached applications** link to access applications that are attached to this policy set and applications that contain attached service resources. To detach a policy set from an application, use the following steps:
   a. In the Filter list, click **Name**.
   b. Find the application. In the Search term(s) field, enter all or part of the name of the application you want to detach and click **Go**.
   c. Detach the application. Find and select the application in the listing and click **Detach Policy Set**.

   To change the policy set that is attached to an application, use the following steps:
   a. In the Filter list, click **Name**.
   b. Find the application. In the Search term(s) field, enter all or part of the name of the application and click **Go**.
   c. Replace the policy set for this application. Find and select the application in the listing and click **Replace Policy Set**. You then have one of the following options:
      - If there are fewer than 25 policy sets to choose from, the Replace Policy Set button is a drop down list and you can click a policy set to attach to the application.
      - If there are more than 25 policy sets to choose from, the Replace Policy Set button opens a collection of policy sets and you can select the policy set to replace from the listing.

Results

Performing these tasks modifies the application to which the custom policy set is attached.

Example

If you had a custom policy set ABC WS-I RSP, for example, that had a misleading description and you want to change the description so that other users know what the policy set really includes, you would access the Policy Set settings view for that policy set. To do this, from the administrative console, you would click **Services > Policy sets > Application policy sets** > ***ABC WS-I RSP***. Then you would click the description field and edit the text so that it better represents the scope of the policy set.

What to do next

If the policy set you have modified is an attached policy set, restart all affected applications to pick up the changes you made. If the policy set is unattached, then no further action is required.

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Related tasks

“Defining and managing policy set bindings” on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

“Creating policy sets using the administrative console” on page 806
You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Related reference

“Search attached applications collection” on page 957
Use this page to search for applications and other resources that are attached to a specific policy set or to search for applications and other resources that have attached service resources.

“Copy of default policy set and bindings settings” on page 815
Use this page to copy a policy set that you select from a list of available policy sets.

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Deleting policy sets using the administrative console

You can use the administrative console to delete the default policy sets or the application specific policy sets that you have created.

Before you begin

If you are deleting an attached policy set, stop all applications and the associated endpoints or operations, then detach and delete the policy set. If the policy set is unattached, then you do not need to detach the policy set before deleting it.

About this task

If a application specific policy set is no longer needed, you might choose to delete it. To delete a application specific policy set, use the Application policy set collection panel.

1. From the administrative console, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**.
2. Click the box in the **Select** column next to the policy set that you want to delete.
3. Click **Delete**.
4. Verify that you want to delete this policy set.

Results

The selected application specific policy set is deleted from the list of policy sets and is no longer available.

Example

You might decide that the *ABC WS-I RSP* policy set that you created did not meet your needs as you had expected and you have already created the *DEF WS-I RSP* policy set to better meet your needs. You have detached all the applications and endpoints from the *ABC WS-I RSP* policy set and you want to delete the policy set. Click **Services > Policy sets > Application policy sets** and find the *ABC WS-I RSP* policy set in the listing. Select the *ABC WS-I RSP* policy set and click **Delete**. Confirm that you want to delete the policy set and it is removed from the listing of policy sets. Restart the applications that you detached the ABC WS-I RSP policy from, and you have successfully deleted the policy set.

Alternatively, and depending on your use case, you might want to reassign the attachment to another resource before deleting the policy set. Make a copy of the policy set that you are about to delete, then attach it later to your application or service artifact. See attaching a policy set to a service artifact. You can now delete the policy set.

What to do next

If you have detached and then deleted a policy set from an application, restart all affected applications to pick up this change.

Related concepts

“Web services policy sets” on page 961
Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Related tasks

“Modifying policy sets using the administrative console” on page 820
You can use the administrative console to modify existing custom policy sets that you have created. If you have copied an existing default policy set or created a policy set yourself, you can always go back and make changes to them to make them better suit the changing needs of your business.

“Creating policy sets using the administrative console” on page 806
You can use the administrative console to either create a policy set by specifying all the necessary information or by copying an existing policy set that you rename. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Attaching a policy set to a service artifact” on page 901
Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

“Deleting policy sets using the wsadmin tool” on page 546
Use the Jython or Jacl scripting language to delete policy sets from your configuration with the wsadmin tool. You must remove all policy set attachments before removing the policy set.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

“Removing policy set attachments using the wsadmin tool” on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Defining and managing policy set bindings

Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

About this task

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific bindings

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and constrained to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have very limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings. General provider policy set bindings might also be used for trust service attachments.

Note: The general bindings that are shipped with the product are provider and client sample bindings. These bindings are initially set as the cell default bindings. Do not use these bindings in their current state in a production environment. To use the sample bindings, modify them to meet your security needs in a production environment. Alternatively, create a copy of the bindings and then modify the copy. For example, change the key and keystore settings to ensure security, and modify other settings to match your environment. You must also configure the username and password for Username or LTPA token authentication. See the topic Configuring the username and password for WS-Security Username or LTPA token authentication for more information.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

To view or work with your current policy sets bindings, perform the following:

1. To view your current policy set and application specific bindings from the administrative console, click **Services > Policy sets > Application policy sets** >*policy_set_name* > **Attached applications** and then click an application. Depending on the application that you select, use one of the following topics for help with the bindings attached to policy sets:
   - Service provider policy sets and bindings
   - Service client policy sets and bindings

   Sort on the **Attached policy set** column on either of the policy sets and bindings pages to select the service resources with the same policy set attached. Likewise, sort on the **Binding** column to select the service resources that share the same custom binding to attach to a different policy set. If you sort on the **Policy Set** or the **Binding** column, the hierarchical relationship of the service resources in the first column is not accurate. You can sort again on the **Application/Service/Endpoint/Operation** column to restore the hierarchical relationship. The entries in the **Application/Service/Endpoint/ Operation** column display in ascending order.
2. To work with an existing bindings from the administrative console, click **Services > Policy sets > Application policy sets** >*policy_set_name* > **Attached applications**. Click an application name, and then click either the **Service provider policy sets and bindings** or the **Service client policy sets and bindings**. Then click a binding name in the Bindings column of the table.

   Note: If no applications appear when you click **Attached applications**, you do not have any applications attached to the selected policy set. To attach a policy set and binding to an application using the administrative console, click **Applications > Enterprise Applications > *application name***. Then, click either **Service provider policy sets and bindings** or **Server client policy sets and bindings** to attach resources to your policy set and to assign bindings.

   .
3. [Optional] To work with general bindings, click **Services > Policy sets > General client policy set bindings** or **Services > Policy sets > General provider policy set bindings**.

   You can complete the following actions for general bindings:
   - “Importing policy set bindings using the administrative console” on page 827
   - “Export policy sets bindings settings” on page 836
   - “Defining and managing service client or provider bindings” on page 831
   - “Creating new or configuring existing general binding settings” on page 835
   - “Copy policy set binding settings” on page 836
   - “Deleting policy set bindings” on page 837

Results

When you finish this task, you would have performed one or more of the following:

- Created an application specific or general policy set binding
- Imported a policy set binding
- Exported a policy set binding
- Deleted a policy set binding
- Modified an application attachment to apply an application specific binding for single security domain
- Modified an application attachment to apply an application specific binding for multiple security domain

Related reference

“Keys and certificates” on page 869
Use this page to link to key and certificate binding configuration panels. This panel defines key and certificate bindings for JAX-WS Web services only. These keys and certificates can be centrally managed by the product or in an external keystore.

“WS-Security authentication and protection” on page 878
Use the links on this page to configure authentication, signature, and encryption information that the policy requires.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

“Service client policy set and bindings collection” on page 763
Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

“Service provider policy sets and bindings collection” on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

Importing policy set bindings using the administrative console

You can import predefined client or provider policy set bindings using the administrative console.

Before you begin

This task only applies to general client or provider bindings. Before you begin this task, review the defining and managing service client or provider bindings.

About this task

The general bindings panel has an Import button to allow the import of client or provider policy set bindings. You can import a policy set binding by specifying the full path and filename of the binding to import.

1. Navigate to the General client policy set bindings or the General provider policy set bindings panel using one of the following ways:
   - **Services > Policy sets > General client policy set bindings**
   - **Services > Policy sets > General provider policy set bindings**
2. Click **Import**.
3. Specify the full path and file name of the policy set binding to import.
4. Select either **Use current binding name** or **Specify a different name to use for this binding**. If you are importing a binding and that binding name already exists in the repository, you must specify a different name to use for the binding import.
5. Click **OK**.

Results

When you finish this task, you have been able to import a policy set binding using the administrative console.

Example

If you have a policy set binding, `XYZ_psbind` and you want to import it, perform the following steps:

1. From the General client or provider policy set bindings panel, click **Import**.
2. Specify the full path and file name of the policy set binding to import or use **Browse** to locate the binding in your file system.
3. Select either **Use current binding name** or **Specify a different name to use for this binding**.
4. Click **OK**.
5. Click **Save**, to save your changes to the configuration.

What to do next

You can export a policy set binding to reuse it.

Related tasks

"Defining and managing service client or provider bindings" on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

"Importing and exporting policy sets to client or server environments using scripting" on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

"Application policy sets collection" on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Import policy sets from a selected location settings" on page 819
Use this page to specify a policy set to import from a selected location.

"Export policy sets bindings settings" on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

Import policy set bindings settings:

Use this page to specify a service client or provider policy set binding to import for your service.

Navigate to the General client policy set bindings or the General provider policy set bindings panel using one of the following ways:

- **Services > Policy sets > General client policy set bindings**
- **Services > Policy sets > General provider policy set bindings**

Click **Import**. Specify a fully qualified path and binding name that you want to use to import the policy set binding. The policy set binding file must be a compressed file.

Related tasks

"Defining and managing service client or provider bindings" on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

"Managing policy sets using the administrative console" on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

"Importing and exporting policy sets to client or server environments using scripting" on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

"Application policy sets collection" on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Import policy sets from default repository settings" on page 818
Use this page to specify a policy set to import from a default repository. You can import predefined policy sets from the default repository.

"Import policy sets from a selected location settings" on page 819
Use this page to specify a policy set to import from a selected location.

"Export policy sets bindings settings" on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

Full Path with file name:

Specify a fully qualified path and file name to the policy set file or use the browse button to locate the file in your file system.

Select **Use current binding name** if you are not changing the binding name and click **OK**.

Select **Specify a different name for this binding** if you are changing the binding name. Provide a new name and click **OK**.

Button	Resulting action
OK	Imports the specified binding file if valid or returns an error if an incorrect file is provided. You are returned to the General client or provider policy set binding panel after the import.
Cancel	Returns you to the General client or provider policy set binding panel without importing a binding.

Web services policy set bindings

A set of bindings is a named object that is associated with a specific policy set and service resource that is attached to the policy set.

Bindings contain environment and platform specific information, like the following types of information:

- Keys used for signature and encryption
- Keystore information

- Authentication information
- Persistent information

Typically, bindings are specific to the application or the platform, and they are not shared.

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and defined by the characteristics of the policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Although general bindings are highly reusable, they do not provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings. General provider policy set bindings might also be used for trust service attachments.

Note:

The sample general bindings that are shipped with the product are provider and client sample. Do not use these sample bindings in their current state in a production environment. However, if they were modified to contain non-sample data, you can use these sample bindings in a production environment.

You cannot assign a binding to a service provider resource that does not have a policy set or has an inherited attachment. To assign a binding to such a service provider resource, you must first attach a policy set to the resource. Also, you cannot assign a binding to a service client resource

that does not have an effective policy configuration or has an inherited policy attachment. To assign a binding to such a service client resource, you must first attach a policy set or specify the use of the provider policy.

Defining and managing service client or provider bindings

Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

Before you begin

You cannot assign a binding to a service provider resource that does not have a policy set or has an inherited attachment. To assign a binding to such a service provider resource, you must first attach a policy set to the resource. Also, you cannot assign a binding to a service client resource that does not have an effective policy configuration or has an inherited policy attachment. To assign a binding to such a service client resource, you must first attach a policy set or specify the use of the provider policy. See attaching a policy set to a service artifact.

About this task

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and defined by the characteristics of the policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in an unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they do not provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

Note: The general bindings that are included with the product are provider and client sample bindings. Do not use these bindings in their current state in a production environment. However, if they were modified to contain non-sample data, they could be used in a production environment.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

You can complete the following tasks to define and manage general client or provider policy set bindings.

1. To create a new general client or provider policy set binding or to manage the binding configuration from the administrative console, click **Services > Policy sets > General client policy set bindings** > **New**. You can also access this panel by clicking **Services > Policy sets > General provider policy set bindings** > **New**. Use the resulting detail panel to create a new client or provider policy set binding. See creating new or configuring existing general binding settings.
2. To copy a specific policy set binding, select the binding name from the table and click **Copy**. See copying a policy set binding settings.
3. To import a client or provider policy set binding, click **Import**. Use the importing policy set bindings using the administrative console to complete the import task that you select.
4. To export a client or provider policy set binding, select the binding name from the table, and click **Export**. See export policy set binding settings.
5. To delete a policy set binding, select the binding name from the table, and click **Delete**. See deleting policy set bindings.

Results

When you finish this task, you have created, copied, exported, imported or deleted a client or provider policy set binding.

Related tasks

"Attaching a policy set to a service artifact" on page 901
Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

"Importing policy set bindings using the administrative console" on page 827
You can import predefined client or provider policy set bindings using the administrative console.

Related reference

"Export policy sets bindings settings" on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

"Copy policy set binding settings" on page 836
Use this page to view and copy general policy set bindings for either a single security or a multiple security domain environment.

"Deleting policy set bindings" on page 837
Use this task to delete general policy set bindings using the administrative console.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Service client or provider policy set bindings collection:

Use this page to create, copy, and manage general policy set bindings, such as the service client or provider bindings. These bindings provide system-specific configuration, and can be reused across policy set attachments. You can select the general default bindings, create new general bindings, or use existing bindings for an attached policy set.

To view this administrative console page, click **Services** → **Policy Sets** → **General client policy set bindings**. You can also view this page by clicking **Services** → **Policy Sets** → **General provider policy set bindings**.

About policy set bindings

In Version 7.0, there are two types of bindings, application specific bindings and general bindings.

Application specific binding

You can create application specific bindings only at a policy set attachment point. These bindings are specific to and defined by the characteristics of the defined policy. Application specific bindings can provide onfiguration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets.

When you create an application specific binding for a policy set attachment, the binding begins in an unconfigured state. You must add each policy, such as WS-Security or HTTP transport, that you want to override the default binding and fully configure the bindings for each policy that you have added. For WS-Security policy, some high level configuration attributes such as TokenConsumer, TokenGenerator, SigningInfo, or EncryptionInfo might be obtained from the default bindings if they are not configured in the application specific bindings.

For service providers, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service provider resources that have an attached policy set. See service providers policy sets and bindings collection. Similarly, for service clients, you can only create application specific bindings by selecting **Assign Binding > New Application Specific Binding** for service client resources that have an attached policy set. See service client policy set and bindings collection.

General bindings

General bindings are new for Version 7.0. These bindings can be configured to be used across a range of policy sets and can be reused across applications and for trust service attachments. Though general bindings are highly reusable, they are however not able to provide configuration for advanced policy requirements, such as multiple signatures. There are two types of general bindings:

- General provider policy set bindings
- General client policy set bindings

You can create general provider policy set bindings by accessing **Services > Policy sets > General provider policy set bindings** > **New** in the general provider policy sets panel or by accessing **Services > Policy sets > General client policy set bindings** > **New** in the general client policy set and bindings panel. See defining and managing service client or provider bindings. General provider policy set bindings might also be used for trust service attachments.

Note:

The sample general bindings that are shipped with the product are provider and client sample. Do not use these sample bindings in their current state in a production environment. However, if they were modified to contain non-sample data, you can use these sample bindings in a production environment.

You cannot assign a binding to a service provider resource that does not have a policy set or has an inherited attachment. To assign a binding to such a service provider resource, you must first attach a policy set to the resource. Also, you cannot assign a binding to a service client resource

that does not have an effective policy configuration or has an inherited policy attachment. To assign a binding to such a service client resource, you must first attach a policy set or specify the use of the provider policy.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

“Defining and managing policy set bindings” on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

“Importing policy set bindings using the administrative console” on page 827
You can import predefined client or provider policy set bindings using the administrative console.

Related reference

“Creating new or configuring existing general binding settings” on page 835
Use this page to create a new client or provider policy set binding. You can also use this page to configure an existing general binding. Empty bindings will be deleted. Scoping a binding to a security domain constrains the configuration options to those applicable to that domain and limits use of the binding to the specified domain.

“Export policy sets bindings settings” on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

“Copy policy set binding settings” on page 836
Use this page to view and copy general policy set bindings for either a single security or a multiple security domain environment.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Name:

Specifies the name of the service client or provider policy set binding.

The following list of buttons are available to create and manage service client or provider policy set bindings:

Button	Resulting Action
New	Creates a new client or provider policy set binding. This option accesses the New binding panel. The binding is empty initially; you must enter a name and an optional description for a new client or provider policy set binding.
Delete	Removes the selected client or provider general policy set binding. The default general bindings are being used as the default for a server or a security domain, including the global security domain. You cannot delete global security default bindings. An attempt to delete such default general binding generates an error message that the selected binding cannot be deleted because it is currently the default binding for the global security domain.
Copy	Creates a modifiable copy of the selected client or provider policy set binding using a new name that you provide.
Import	Imports a client or provider policy set binding. This is a menu item with the option of importing a binding with the same name or providing a different name for the import.
Export	Exports the selected client or provider policy set binding to an archive file.

Description:

Specifies the user-defined description for the client or provider policy set bindings.

Creating new or configuring existing general binding settings:

Use this page to create a new client or provider policy set binding. You can also use this page to configure an existing general binding. Empty bindings will be deleted. Scoping a binding to a security domain constrains the configuration options to those applicable to that domain and limits use of the binding to the specified domain.

To view this administrative console page, click **Services > Policy sets > General client policy set bindings** > **New**. You can also view this page by clicking **Services > Policy sets > General provider policy set bindings** > **New**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Related tasks

"Defining and managing policy set bindings" on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

"Importing policy set bindings using the administrative console" on page 827
You can import predefined client or provider policy set bindings using the administrative console.

"Deleting policy set bindings" on page 837
Use this task to delete general policy set bindings using the administrative console.

Related reference

"Export policy sets bindings settings" on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

"Copy policy set binding settings" on page 836
Use this page to view and copy general policy set bindings for either a single security or a multiple security domain environment.

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Bindings configuration name:

Specifies the name of the new binding configuration.

You must enter a binding configuration name to proceed. The name of the binding is automatically applied when you add a policy type and provide additional configuration information. At that time, the initial **Cancel** button is replaced with an **OK** button. Each binding that you add is initially empty, and you are directed to the binding configuration panel to provide additional information for the binding.

Button	Resulting Action
Add	Adds a policy type, such as WS-Security, to a new binding that you create or to an existing binding. You can access the configuration of each policy type binding through the policy type name link. If you click **Cancel** without providing any configuration information for the policy type, that policy type is not saved or included in the list of policy types. You might receive an attention message that the policy type was not saved as part of the binding configuration.
Delete	Deletes a policy type from the list.

Button	Resulting Action
OK	Associates the name of the binding to the policy type information after you add a policy type and provide additional configuration information.
Cancel	Returns you to the General client or provider policy set bindings panel without creating a new binding configuration.

Security domain:

This field is only available in a multiple security domain environment and specifies a valid security domain. You can scope the binding to that particular security domain, which is scoped to the global security domain by default.

Description:

Specifies a brief description of the new client or provider policy set binding.

Export policy sets bindings settings

This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

Navigate to the General client policy set bindings or the General provider policy set bindings panel using one of the following paths:

- **Services > Policy sets > General client policy set bindings**
- **Services > Policy sets > General provider policy set bindings**

1. Select a binding from the list and click **Export**.
2. Click the binding to download the archive file.
3. Use the file dialog box to select a location for the exported binding. The policy set binding file is exported as a compressed file.
4. Click **Done**.

Related tasks

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Importing and exporting policy sets to client or server environments using scripting” on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Import policy sets from default repository settings” on page 818
Use this page to specify a policy set to import from a default repository. You can import predefined policy sets from the default repository.

Copy policy set binding settings

Use this page to view and copy general policy set bindings for either a single security or a multiple security domain environment.

To access this administrative console page, perform the following steps:

1. Click **Services > Policy sets > General client policy set bindings**. You can also access this page by clicking **Services > Policy sets > General provider policy set bindings**.
2. Select an existing binding from the list, and click **Copy**.
3. Provide a name and description for the copy.
4. Click **OK**.
5. Click **Save**, to save your changes.

Related tasks

"Importing policy set bindings using the administrative console" on page 827
You can import predefined client or provider policy set bindings using the administrative console.

Related reference

"Creating new or configuring existing general binding settings" on page 835
Use this page to create a new client or provider policy set binding. You can also use this page to configure an existing general binding. Empty bindings will be deleted. Scoping a binding to a security domain constrains the configuration options to those applicable to that domain and limits use of the binding to the specified domain.

"Export policy sets bindings settings" on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

Name:

Specifies the name of copy. You must provide a different name for the copy of the binding configuration.

Security domain:

Specifies a valid security domain. This field is only available in a multiple security domain environment. You can scope the binding to a particular security domain; it is scoped to the global security domain by default.

Description:

Specifies a brief description of the copy of the policy set binding.

Deleting policy set bindings

Use this task to delete general policy set bindings using the administrative console.

About this task

You cannot delete general bindings specified as the default for any server or domain. You also cannot delete general bindings that are assigned to a policy set attachment. To delete other bindings, perform the following steps:

1. Click **Services > Policy sets > General client policy set bindings**. You can also access this page by clicking **Services > Policy sets > General provider policy set bindings**.
2. Select an existing binding from the list, and click **Delete**.
3. Click **Save**, to save your changes.

Results

When you finish this task, you have deleted a general policy set binding.

Related tasks

“Importing policy set bindings using the administrative console” on page 827
You can import predefined client or provider policy set bindings using the administrative console.

Related reference

“Service client or provider policy set bindings collection” on page 832
Use this page to create, copy, and manage general policy set bindings, such as the service client or provider bindings. These bindings provide system-specific configuration, and can be reused across policy set attachments. You can select the general default bindings, create new general bindings, or use existing bindings for an attached policy set.

“Creating new or configuring existing general binding settings” on page 835
Use this page to create a new client or provider policy set binding. You can also use this page to configure an existing general binding. Empty bindings will be deleted. Scoping a binding to a security domain constrains the configuration options to those applicable to that domain and limits use of the binding to the specified domain.

“Export policy sets bindings settings” on page 836
This task only applies to general client or provider bindings. Use this page to export either a client or provider policy set binding for reuse.

“Copy policy set binding settings” on page 836
Use this page to view and copy general policy set bindings for either a single security or a multiple security domain environment.

Creating application specific bindings for policy set attachment

After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

Before you begin

Before you can start this task, you must deploy an application containing Web services, and attach a policy set. See attaching a policy set to a service artifact.

About this task

The application specific binding panels display options based on the definitions in the attached policy set. For example, if the policy set does not have WS-ReliableMessaging configured, then the application specific binding panels do not display an option to configure WS-ReliableMessaging bindings. To create application specific bindings for an attached policy set, perform the following steps:

1. Open the administrative console.
2. To create application specific bindings for a service provider policy attachment, click **Applications > Application Types > WebSphere enterprise applications** >*Service_provider_application_instance* > **Service provider policy sets and bindings**.

   To create application specific bindings for a service client policy attachment, click **Applications > Enterprise applications** > ***application_name*** > **Service client policy sets and bindings**.
3. Select the check box for the service artifact with an attached policy set.
4. Click **Assign Binding** to see a list of available bindings to assign.
5. [Optional] Select a previously defined application specific attachment binding from the list.

   If you decide to create a new application specific binding, complete steps 6 through 9.
6. Click **New**.
7. Specify a name for the binding and select which policies that you want to include in this application specific binding. Click **Add**, and select the policies to add.
8. Complete the information for the specified policies to define associated configuration data.
9. Click **Save**, to save your changes to the master configuration.

Results

When you finish this task, a application specific binding is created for the service artifact with the attached policy set.

Example

You have the application, `app1` with the attached policy set `Username WSSecurity default`, and you want to define application specific bindings. First locate the application in the **Applications > Application Types > WebSphere enterprise applications** collection. Click the **app1** application. You can then click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link. Select the check box for the `app1` application where the policy set is attached. Click **Assign Binding** and click **New**. Specify a name for the binding, such as `myBinding`. Click **Add** and select **WS-Security** from the list. Next, click the link for **WS-Security** policy and complete the panels with the appropriate information. Click **Save**, to save your changes to the master configuration.

What to do next

Restart the application that contains the service artifact with the application specific bindings you created.

Related tasks

"Attaching a policy set to a service artifact" on page 901
Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

"Defining and managing service client or provider bindings" on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

Related reference

"Creating new or configuring existing general binding settings" on page 835
Use this page to create a new client or provider policy set binding. You can also use this page to configure an existing general binding. Empty bindings will be deleted. Scoping a binding to a security domain constrains the configuration options to those applicable to that domain and limits use of the binding to the specified domain.

Setting server default bindings for policy sets

You can set server default bindings if you want the policy set attachments for service providers and clients that are deployed to the server to use bindings that are different than those that are specified for the cell. If you use multiple security domains, your default server bindings will also override the security domain default bindings.

Before you begin

Before you can set server default bindings for your Java API for XML-Based Web Services (JAX-WS) application, you must first configure at least one general provider policy set binding or general client policy set binding. To define and manage these general bindings, use the administrative console and select **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings** .

About this task

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product.

In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general bindings that are used as the default for an application server or a security domain.

General service provider and client bindings are not linked to a particular policy set, and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings, and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers.

You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The following list is the order of precedence from lowest to highest that the application server uses to determine which default bindings to use:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided **Provider sample** as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided **Client sample** as the default service client binding, you can select an existing general client binding or create a new general client binding. To specify your global security (cell) default bindings, use the administrative console, and click **Services > Policy sets > Default policy set bindings**. For environments with multiple security domains, you can optionally choose the general provider and general client bindings that you want to use as the default bindings for a domain.

In addition to choosing default bindings for the global security (cell), you can also choose the general provider and general client bindings that you want to use as the default bindings for a server. This is only necessary if you want to use different default bindings for a particular server than those used by the other servers in the security domain or cell.

To choose the default bindings for a server from the administrative console, click **Servers > Server Types > WebSphere application servers > *server_name*** and then under Security, click **Default policy set bindings**. If you do not choose a general binding as the default for a server, the default bindings for the domain in which the server resides is used. If you do not choose a binding as the default for a domain, the default bindings for the global security (cell) are used. You must choose a default service provider and default service client bindings for the cell. The general bindings that are included with the product are initially set as the global security (cell) default bindings. You cannot delete a binding that is used as part of any policy set attachment or specified as the default binding for server, a domain, or the cell. To learn more about defining default bindings for a server, see the server default bindings documentation.

Note:

If you have an application that contains one or more application specific bindings that are configured at the WebSphere Application Server V6.1 level, this application is a V6.1 application. If you have applications that are deployed to Version 6.1 servers within the Version 7.0 application

server environment or you have V6.1 applications that are deployed to V7.0 application servers, you can specify Version 6.1 default policy set bindings for the cell. These bindings are used for both client and provider policy set attachments within V6.1 applications and attachments to service applications that are deployed to a V6.1 server. Additionally, these default bindings are used for V6.1 attachments unless they are overridden at the attachment point by an application specific binding or a V6.1 server default binding. You can upgrade V6.1 bindings to WebSphere Application Server V7.0 bindings for by using the upgradeBindings command using the wsadmin tool, if the V6.1 application is not installed on WebSphere Application Server V6.1.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. To set default policy set bindings for your server, select**Servers > Server Types > WebSphere application servers > *server_name* > Default policy set bindings**.
3. Select the server default provider binding.

   If you specify a server default provider binding, the selected binding overrides the default provider bindings that are specified for the cell or the security domain to which the server is deployed. The default setting is **None**.

   If multiple security domains are in use, the name of the security domain to which each binding is scoped displays beside the name of each available provider binding. Only the bindings that are scoped to the global security level or to the security domain to which the server is deployed are displayed.
4. Select the server default client binding.

   If you specify a server default client binding, the selected binding overrides the default client bindings that are specified for the cell or the security domain to which the server is deployed. The default setting is **None**.

   If multiple security domains are in use, the name of the security domain to which each binding is scoped displays beside the name of each available provider binding. Only the bindings that are scoped to the global security level or to the security domain to which the server is deployed are displayed.
5. Click **Apply** or **OK** to submit your changes.
6. Click **Save** to save your changes to the master configuration.
7. (optional) If you are using a Version 6.1 application, you can specify server V6.1 default policy set bindings. To set these bindings, select **Servers > Server Types > WebSphere application servers > *server_name* > Default policy set bindings > Version 6.1 default policy set bindings**.

   Note: Select the V6.1 default bindings for this server. If your application contains one or more application specific bindings that are configured at the WebSphere Application Server V6.1 level, this application is a V6.1 application. These default bindings are used for client and provider policy set attachments for applications that are deployed to V6.1 servers, and for V6.1 applications that are deployed to V7.0 servers. These bindings are used for both client and provider policy set attachments within V6.1 applications and attachments to service applications that are deployed to a V6.1 server. Additionally, these default bindings are used for V6.1 attachments unless they are overridden at the attachment point by an application specific binding or a V6.1 server default binding.

Results

When you complete these steps, the server default bindings are defined and all policy set attachments that specify use of the default binding for your Web service applications that are deployed to the server will use server level default bindings.

Example

Suppose you have configured an application server, `server1`, and you have deployed several Web service applications to the `server1` application server. Because these applications have similar security and quality of service requirements and you plan for them to share security configuration, you want to define the default bindings for policy set attachments to service providers and clients using the `server1`.

Suppose also that you want to modify the provided general provider binding, **Provider sample**. You can copy and modify this provided sample to take advantage of existing bindings.

1. Copy and modify the provided **Provider sample** and **Client sample** to meet your security and quality of service requirements. Include binding configuration for all policy types.
   - Click **Services > Policy sets > General provider policy set bindings**. Select **Provider sample > copy**. Name the new general provider binding, MyServiceProviderbinding , and provide a description for the new binding.
   - Click **Services > Policy sets > General client policy set bindings**. Select **Client sample > copy**. Name the new general client binding, MyServiceClientbinding, and provide a description for the new binding.
2. Locate *server1* in the Application servers collection and click the instance. From the administrative console, select **Servers > Server Types > WebSphere application servers** , and click the *server1* instance.
3. Click **Default policy set bindings**.
4. Select the bindings that you want to use for your provider and client policy set attachments. In this example, select your customized general bindings, MyServiceProviderbinding and MyServiceClientbinding.
5. Click **Apply** or **OK** to submit your changes.
6. Click **Save** to save your changes to the master configuration.

Each time you attach a policy set to a service or client deployed to the *server1* application server, it is initially set to use the specified bindings.

.

What to do next

After setting server default bindings, you can start deploying services to the server and start attaching policy sets. Alternatively, you might already have services deployed to the server, and the server is using the global default bindings because there is no server default binding. Now that you have set server default bindings, ensure that the server default bindings are used for the service messages as specified.

Related tasks

“Attaching a policy set to a service artifact” on page 901
Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

“Setting default policy set bindings” on page 846
You can set provider and client default policy set bindings that are used as the global security default policy set bindings. The specified global security default bindings apply to all Web services unless the bindings are overridden at the attachment point, at the server, or at a security domain.

“Default policy set bindings collection” on page 850
Use this page to specify the service provider and client default bindings. The specified service provider and client bindings are used at the cell (global security) level unless these specified bindings are overridden at the attachment point, at the server, or at a security domain.

Related reference

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Server default binding settings:

Use this page to specify the server default bindings if you want to override the default bindings that are specified for the cell (global security) or the security domain to which the server is deployed.

To view this administrative console page, complete the following actions:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***.
2. From Security, click **Default policy set bindings**.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Policy set bindings for servers

To understand default policy set bindings, it is important to first understand policy set bindings.

Policy set bindings contain platform-specific information, such as keystore, authentication information or persistent information that is required by a policy set attachment. A policy set attachment is a policy set that is attached to an application resource. In WebSphere Application Server Version 7.0, there are two types of bindings, general bindings and application specific bindings.

There are two types of general bindings, general service provider bindings and general service client bindings. You can configure one or more general service provider bindings and one or more general service client bindings across a range of policy sets. Additionally, you can re-use these general bindings across applications and for trust service attachments. To define and manage general bindings, in the administrative console click **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**. The general service provider and client bindings have independent settings that you can customize to meet the needs of your environment. To learn more about general bindings, read about defining and managing policy set bindings.

You can create application specific bindings when you attach a policy set to an application resource. These bindings are specific to and defined to the characteristics of the policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets. To assign application specific bindings to an application for service providers, in the administrative console click **Applications > Applications Types > WebSphere**

enterprise applications > *application_name*> **Service provider policy sets and bindings > Assign Binding > New Application Specific Binding**. To assign application specific bindings to an application for service clients, in the administrative console click **Applications > Applications Types > WebSphere enterprise applications >** *application_name*> **Service client policy sets and bindings > Assign Binding > New Application Specific Binding**.

Default policy set bindings for servers

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default cell (global security) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings.

General service provider and client bindings are not linked to a particular policy set, and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by assigning a security domain with a default binding to your server.

You can specify default bindings for your service provider or client that are used at the cell (global security) level, for a security domain, or for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The application server uses the following order of precedence, from lowest to highest, when determining which default bindings to use:

1. Server level default
2. Security domain level default
3. Cell (global security) default

The general bindings that are provided with the product are initially set as the cell (global security) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. If you do not want to use the provided **Provider sample** as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided **Client sample** as the default service client binding, you can select an existing general client binding or create a new general client binding. To specify a cell (global security) default bindings, in the administrative console click **Services > Policy sets > Default policy set bindings**. For environments with multiple security domains, you can optionally choose the general provider and general client bindings that you want to use as the default bindings for a domain. To learn more about default policy set bindings for the cell (global security), see the default policy set bindings documentation.

In addition to choosing default bindings for the cell (global security), you can also choose the general provider and general client bindings that you want to use as the default bindings for a server. Use this page to choose the default bindings for a server from the administrative console. Click **Servers > Server Types > WebSphere application servers >** *server_name* and then from Security, click **Default policy set bindings**. If you do not choose a general binding as the default for a server, the default bindings for the domain in which the server resides is used. If you do not choose a binding as the default for a domain, the default bindings for the cell (global security) is used. You must choose a default service provider and default service client bindings for the cell. The general bindings that are included with the product are initially set as the cell (global security) default bindings. You cannot delete a binding that has been

selected as the default binding for server, a domain, or the cell. Before you delete the binding, you must select a different binding as the default or choose to use the defaults for the cell (global security).

Note:

If you have an application that contains one or more application specific bindings that are configured at the WebSphere Application Server V6.1 level, this application is a V6.1 application. If you have applications that are deployed to Version 6.1 servers within the Version 7.0 application server environment or you have V6.1 applications that are deployed to V7.0 application servers, you can specify Version 6.1 default policy set bindings for the cell. These bindings are used for both client and provider policy set attachments within V6.1 applications and attachments to service applications that are deployed to a V6.1 server. Additionally, these default bindings are used for V6.1 attachments unless they are overridden at the attachment point by an application specific binding or a V6.1 server default binding. You can upgrade V6.1 bindings to WebSphere Application Server V7.0 bindings for by using the upgradeBindings command using the wsadmin tool, if the V6.1 application is not installed on WebSphere Application Server V6.1.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Server default provider binding:

Specifies the default server binding. Select the name of the binding you want to use as the default for service providers deployed to this server.

If you are using multiple security domains, the name of the security domain to which the binding is scoped is specified with the name of the server default provider binding. Only bindings that are scoped to global security or to the security domain to which the server is assigned are available in the list.

Note: It is a best practice to specify a default binding that includes all of the policy types. This practice ensures that your default service provider binding has the necessary configuration for all policy types that you want to use.

Server default client binding:

Specifies the default client binding. Select the name of the binding you want to use as the default for service clients deployed to this server.

If you are using multiple security domains, the name of the security domain to which the binding is scoped is specified with the name of the server default client binding. Only bindings that are scoped to global security or to the security domain to which the server is assigned are available in the list.

Note: It is a best practice to specify a default binding that includes all of the policy types. This practice ensures that your default service client binding has the necessary configuration for all policy types that you want to use.

Version 6.1 default bindings:

If you have a Version 6.1 application that you want to use in the Version 7.0 application server environment and you want to specify Version 6.1 default bindings for this server, you can click this link to specify Version 6.1 default bindings.

Server Version 6.1 default policy set bindings:

Use this page to specify the server Version 6.1 default policy set bindings for this server. These default bindings are used for client and provider policy set attachments for applications that are deployed to Version 6.1 servers, and for Version 6.1 applications that are deployed to Version 7.0 servers. The default bindings are used for Version 6.1 attachments unless overridden at the attachment point.

To view this administrative console page, complete the following actions:

1. Click **Servers > Server Types > WebSphere application servers >** ***server_name***
2. Under Security, click **Default policy set bindings**
3. Click **Version 6.1 default policy set bindings**

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Version 6.1 default policy set bindings

You can install a WebSphere Application Server Version 6.1 Feature Pack for Web Services application within the WebSphere Application Server Version 7.0 environment. If your application contains Version 6.1 custom bindings, the application is defined as a Version 6.1 application to the application server. Additionally, if your application is installed on a WebSphere Application Server Version 6.1 Feature Pack for Web Services server within the WebSphere Application Server Version 7.0 environment, the custom binding is created as a Version 6.1 custom binding.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Bindings configuration name:

Specifies the name of the policy set bindings configuration. This field is read-only for default policy set bindings.

Button	Resulting action
Add	Adds a policy to the collection.
Delete	Removes a policy from the collection.

Setting default policy set bindings

You can set provider and client default policy set bindings that are used as the global security default policy set bindings. The specified global security default bindings apply to all Web services unless the bindings are overridden at the attachment point, at the server, or at a security domain.

Before you begin

You must first install and configure an application server. After the application server is installed, you must install a Java API for XML-Based Web Services (JAX-WS) application onto your server. Now, you are ready to attach a policy set to the Web service application. You can define and manage general service provider and client policy set bindings for your Web service resource by using the administrative console.

About this task

Policy set bindings for servers

After you understand policy set bindings, then it is easier to understand how the default bindings are used.

Policy set bindings contain platform-specific information, such as keystore, authentication information or persistent information that is required by a policy set attachment. A policy set attachment is a policy set

that is attached to an application resource. In WebSphere Application Server Version 7.0, there are two types of bindings, general bindings and application specific bindings.

There are two types of general bindings, *general service provider bindings* and *general service client bindings*. You can configure one or more general service provider bindings and one or more general service client bindings and then use them across a range of policy sets. Additionally, you can re-use these general bindings across applications and for trust service attachments. To define and manage general bindings, in the administrative console click **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**. The general service provider and client bindings have independent settings that you can customize to meet the needs of your environment.

You can create *application specific bindings* when you attach a policy set to a Web service application resource. These bindings are specific to and defined by to the characteristics of the defined policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets. To assign application specific bindings to an application for service providers, in the administrative console click **Applications > Applications Types > WebSphere enterprise applications > *application_name* > Service provider policy sets and bindings**. Select a Web service resource with an attached policy and click **Assign Binding > New Application Specific Binding**. To assign application specific bindings to an application for service clients, in the administrative console click **Applications > Applications Types > WebSphere enterprise applications > *application_name* > Service client policy sets and bindings** . Select a Web services resource with an attached policy and click **Assign Binding > New Application Specific Binding**. You can additionally assign application specific bindings for service providers or service clients using the administrative console and click **Services > Service providers > *application_name*** or **Services > Service clients > *application_name*** and then select a Web services resource with an attached policy and assign your bindings.

To learn more about general bindings or application specific bindings, read about defining and managing policy set bindings.

Default policy set bindings

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default global security (cell) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings. After you have configured general bindings, you can specify which of these bindings is the global default binding. You can also optionally specify general bindings that are used as the default for an application server or a security domain.

General service provider and client bindings are not linked to a particular policy set, and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding type to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also share binding configuration by either assigning the binding to each application that is deployed to the server or by setting default bindings for a security domain and assigning the security domain to one or more servers.

You can specify default bindings for your service provider or client that are used at the global security (cell) level, for a security domain, for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The following list is the order of precedence from lowest to highest that the application server uses to determine which default bindings to use:

1. Server level default
2. Security domain level default
3. Global security (cell) default

The sample general bindings that are provided with the product are initially set as the global security (cell) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. For trust service attachments, the default bindings are used when no trust specific bindings are assigned. If you do not want to use the provided **Provider sample** as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided **Client sample** as the default service client binding, you can select an existing general client binding or create a new general client binding. To specify your global security (cell) default bindings, in the administrative console click **Services > Policy sets > Default policy set bindings**. For environments with multiple security domains, you can optionally choose the general provider and general client bindings that you want to use as the default bindings for a domain.

In addition to choosing default bindings for the global security (cell), you can also choose the general provider and general client bindings that you want to use as the default bindings for a server. This is only necessary if you want to use different default bindings for a particular server than those used by the other servers in the security domain or cell. To choose the default bindings for a server from the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name* and then from Security, click **Default policy set bindings**. If you do not choose a general binding as the default for a server, the default bindings for the domain in which the server resides is used. If you do not choose a binding as the default for a domain, the default bindings for the global security (cell) are used. You must choose a default service provider and default service client bindings for the cell. The general bindings that are included with the product are initially set as the global security (cell) default bindings. You cannot delete a binding that is used as part of any policy set attachment or specified as the default binding for the server, a domain, or the cell. To learn more about defining default bindings for a server, see the server default bindings documentation.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Open the administrative console.
2. To set global security default policy set bindings, select **Services > Policy sets > Default policy set bindings**.
3. Select the default service provider binding. The default service provider binding is used as the default for policy set attachments unless the provider or client binding is overridden at the attachment point, at the server, or at a security domain. The default setting is **Provider sample**.
4. Select the default service client binding. If you specify a default service client binding, the selected binding overrides the default bindings that are specified for the cell or the security domain to which the server is deployed. The default setting is **Client sample**.
5. If multiple security domains are enabled, you can view the default provider bindings and the default client bindings for each security domain that is defined in the security domain default bindings collection. You can select the security domain name link if you want to access the domain and select different default bindings. Additionally, you can select the default provider binding links or the default client binding links to access the default bindings and select different default binding settings.
6. Click **Apply** to apply selected bindings as the global default bindings.
7. Click **Save** to save your changes to the master configuration.
8. (optional) If you are using a Version 6.1 application, you can specify server V6.1 default policy set bindings. To set these bindings, select **Services > Policy sets > Default policy set bindings > Version 6.1 default policy set bindings**.

Note:

If you have an application that contains one or more application specific bindings that are configured at the WebSphere Application Server V6.1 level, this application is a V6.1 application. If you have applications that are deployed to Version 6.1 servers within the Version 7.0 application server environment or you have V6.1 applications that are deployed to V7.0 application servers, you can specify Version 6.1 default policy set bindings for the cell. These bindings are used for both client and provider policy set attachments within V6.1 applications and attachments to service applications that are deployed to a V6.1 server. Additionally, these default bindings are used for V6.1 attachments unless they are overridden at the attachment point by an application specific binding or a V6.1 server default binding. You can upgrade V6.1 bindings to WebSphere Application Server V7.0 bindings for by using the upgradeBindings command using the wsadmin tool, if the V6.1 application is not installed on WebSphere Application Server V6.1.

Results

When you complete these steps, you have defined your global security (cell) default policy set bindings and domain default policy set bindings, if applicable.

Example

Suppose that you do not want to use the provided general provider binding, **Provider sample**, as your default service provider binding. To take advantage of existing bindings, you can copy and modify the **Provider sample** to meet your business needs. This example assumes that your server environment has SecurityDomain1 and SecurityDomain2 defined.

1. Copy and modify the provided **Provider sample** general service provider binding. Click **Services > Policy sets > General provider policy set bindings**. Select **Provider sample > copy**. Name the new general provider binding, MyServiceProviderbinding, and provide a description for the new binding.
2. Copy and modify the provided **Client sample** general service client binding. Click **Services > Policy sets > General client policy set bindings**. Select **Client sample > copy**. Name the new general client binding, MyServiceClientbinding, and provide a description for the new binding.
3. To specify the default policy set bindings for your global security (cell) and for your domains, click **Services > Policy sets > Default policy set bindings**. From this page, select MyServiceProviderbinding as the default service provider binding, and select MyClientProviderbinding as the default service client binding.
4. Click **Apply** and **Save** to save your changes to the master configuration.

Assigning a domain default binding is optional. Generally, you assign domain default policy set bindings only when you want the servers in the domain to use different default bindings than the rest of the cell. In this example, suppose you have defined another general provider binding, MyServiceProviderbinding2, and you want to specify this binding as the domain default binding for your SecurityDomain1 domain.

1. From the security domain default bindings collection click **SecurityDomain1 > Default policy set bindings**. From this page, you can select MyServiceProviderbinding2 as the service provider domain default binding.
2. Click **Apply** and **Save** to save your changes to the master configuration.

Related tasks

“Managing policy sets using the administrative console” on page 803
You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units. You can use the administrative console to create, modify, and delete custom policy sets.

“Server default binding settings” on page 843
Use this page to specify the server default bindings if you want to override the default bindings that are specified for the cell (global security) or the security domain to which the server is deployed.

“Setting server default bindings for policy sets” on page 839
You can set server default bindings if you want the policy set attachments for service providers and clients that are deployed to the server to use bindings that are different than those that are specified for the cell. If you use multiple security domains, your default server bindings will also override the security domain default bindings.

Related reference

Administrative roles
The Java Platform, Enterprise Edition (Java EE) role-based authorization concept is extended to protect the WebSphere Application Server administrative subsystem.

Default policy set bindings collection:

Use this page to specify the service provider and client default bindings. The specified service provider and client bindings are used at the cell (global security) level unless these specified bindings are overridden at the attachment point, at the server, or at a security domain.

To view this administrative console page, click **Services > Policy sets > Default policy set bindings**.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Policy set bindings for servers

To understand default policy set bindings, it is important to first understand policy set bindings.

Policy set bindings contain platform-specific information, such as keystore, authentication information or persistent information that is required by a policy set attachment. A policy set attachment is a policy set that is attached to an application resource. In WebSphere Application Server Version 7.0, there are two types of bindings, general bindings and application specific bindings.

There are two types of general bindings, general service provider bindings and general service client bindings. You can configure one or more general service provider bindings and one or more general service client bindings across a range of policy sets. Additionally, you can re-use these general bindings across applications and for trust service attachments. To define and manage general bindings, in the administrative console click **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**. The general service provider and client bindings have independent settings that you can customize to meet the needs of your environment. To learn more about general bindings, read about defining and managing policy set bindings.

You can create application specific bindings when you attach a policy set to an application resource. These bindings are specific to and defined to the characteristics of the policy. Application specific bindings are capable of providing configuration for advanced policy requirements, such as multiple signatures; however, these bindings are only reusable within an application. Furthermore, application specific bindings have limited reuse across policy sets. To assign application specific bindings to an application for service providers, in the administrative console click **Applications > Applications Types > WebSphere enterprise applications >** *application_name*> **Service provider policy sets and bindings > Assign Binding > New Application Specific Binding**. To assign application specific bindings to an application for

service clients, in the administrative console click **Applications > Applications Types > WebSphere enterprise applications >** *application_name*> **Service client policy sets and bindings > Assign Binding > New Application Specific Binding**.

Default policy set bindings for servers

Note: In WebSphere Application Server Version 7.0, the security model is enhanced to a domain-centric security model instead of a server-based security model. The configuration of the default cell (global security) level and default server level bindings has also changed in this version of the product. In the WebSphere Application Server Version 6.1 Feature Pack for Web Services, you can configure one set of default bindings for the cell and optionally configure one set of default bindings for each server. In Version 7.0, you can configure one or more general service provider bindings and one or more general service client bindings.

General service provider and client bindings are not linked to a particular policy set, and they provide configuration information that you can reuse across multiple applications. You can create and manage general provider and client policy set bindings and then select one of each binding to use as the default for an application server. Setting the server default bindings is useful if you want the services that are deployed to a server to share binding configuration. You can also accomplish this sharing of binding configuration by assigning the binding to each application deployed to the server or by assigning a security domain with a default binding to your server.

You can specify default bindings for your service provider or client that are used at the cell (global security) level, for a security domain, or for a particular server. The default bindings are used in the absence of an overriding binding specified at a lower scope. The application server uses the following order of precedence, from lowest to highest, when determining which default bindings to use:

1. Server level default
2. Security domain level default
3. Cell (global security) default

The general bindings that are provided with the product are initially set as the cell (global security) default bindings. The default service provider binding and the default service client bindings are used when no application specific bindings or trust service bindings are assigned to a policy set attachment. If you do not want to use the provided **Provider sample** as the default service provider binding, you can select an existing general provider binding or create a new general provider binding to meet your business needs. Likewise, if you do not want to use the provided **Client sample** as the default service client binding, you can select an existing general client binding or create a new general client binding. To specify a cell (global security) default bindings, in the administrative console click **Services > Policy sets > Default policy set bindings**. For environments with multiple security domains, you can optionally choose the general provider and general client bindings that you want to use as the default bindings for a domain. To learn more about default policy set bindings for the cell (global security), see the default policy set bindings documentation.

In addition to choosing default bindings for the cell (global security), you can also choose the general provider and general client bindings that you want to use as the default bindings for a server. Use this page to choose the default bindings for a server from the administrative console. Click **Servers > Server Types > WebSphere application servers >** *server_name* and then from Security, click **Default policy set bindings**. If you do not choose a general binding as the default for a server, the default bindings for the domain in which the server resides is used. If you do not choose a binding as the default for a domain, the default bindings for the cell (global security) is used. You must choose a default service provider and default service client bindings for the cell. The general bindings that are included with the product are initially set as the cell (global security) default bindings. You cannot delete a binding that has been selected as the default binding for server, a domain, or the cell. Before you delete the binding, you must select a different binding as the default or choose to use the defaults for the cell (global security).

Note:

If you have an application that contains one or more application specific bindings that are configured at the WebSphere Application Server V6.1 level, this application is a V6.1 application. If you have applications that are deployed to Version 6.1 servers within the Version 7.0 application server environment or you have V6.1 applications that are deployed to V7.0 application servers, you can specify Version 6.1 default policy set bindings for the cell. These bindings are used for both client and provider policy set attachments within V6.1 applications and attachments to service applications that are deployed to a V6.1 server. Additionally, these default bindings are used for V6.1 attachments unless they are overridden at the attachment point by an application specific binding or a V6.1 server default binding. You can upgrade V6.1 bindings to WebSphere Application Server V7.0 bindings for by using the upgradeBindings command using the wsadmin tool, if the V6.1 application is not installed on WebSphere Application Server V6.1.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Default service provider binding:

Specifies the default service provider binding that is used as the default for policy set attachments. You can override this default at the attachment point or by a lower level default.

Note: It is a best practice to specify a default binding that includes all of the policy types. This practice ensures that your default service provider binding has the necessary configuration for all policy types that you want to use..

Default service client binding:

Specifies the default service client binding that is used as the default for policy set attachments. You can override this default at the attachment point or by a lower level default.

Note: It is a best practice to specify a default binding that includes all of the policy types. This practice ensures that your default service client binding has the necessary configuration for all policy types that you want to use.

Security domain default bindings:

Specifies a collection of security domain default bindings. This collection only displays if you are using multiple security domains.

If you are using multiple security domains, this collection displays the default client and provider bindings for each security domain. You can select the security domain name link if you want to access the domain and select different default bindings.

Security domain:

Specifies the name of a security domain.

Default provider binding:

Specifies the default service provider binding for the security domain. You can view the default provider binding settings by clicking on the name of default provider binding.

Default client Binding:

Specifies the default client binding for the security domain. You can view the default client binding settings by clicking on the name of default client binding.

Version 6.1 default bindings:

If you have a Version 6.1 application that you want to use in the Version 7.0 application server environment and you want to specify Version 6.1 default bindings for this server, you can click this link to specify Version 6.1 default bindings.

Version 6.1 default policy set bindings:

Use this page to specify Version 6.1 default policy set bindings for the cell (global security). These bindings are used for both client and provider policy set attachments within Version 6.1 applications and attachments to service applications that are deployed to a Version 6.1 server. These default bindings are used for Version 6.1 attachments unless they are overridden at the attachment point or by a Version 6.1 server default binding.

To view this administrative console page when you are specifying Version 6.1 cell level default policy set bindings, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**
2. Click **Version 6.1 default policy set bindings**

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Version 6.1 default policy set bindings

You can install a WebSphere Application Server Version 6.1 Feature Pack for Web Services application within the WebSphere Application Server Version 7.0 environment. If your application contains Version 6.1 custom bindings, the application is defined as a V6.1 application to the application server. Additionally, if your application is installed on a WebSphere Application Server Version 6.1 Feature Pack for Web Services server within the WebSphere Application Server Version 7.0 environment, the custom binding is created as a V6.1 custom binding.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Bindings configuration name:

Specifies the name of the policy set bindings configuration. This field is read-only for default policy set bindings.

Policies – HTTP transport:

Links to the HTTP transport policy settings page where you define the HTTP transport settings. The HTTP features and HTTP connection polices are applied to outbound messages. The response listener policy is enforced on inbound messages.

Policies – SSL transport:

Links to the SSL transport policy configuration settings page where you define the SSL transport settings.

Policies – WS-Addressing:

Links to the configuration settings page for the WS-Addressing policy. In a network deployment environment, use this page to enable or disable workload management. Otherwise, you can attach the WS-Addressing policy set to service resources. No additional configuration is required.

Policies – WS-ReliableMessaging:

Links to the page where you configure the WS-ReliableMessaging bindings.

Policies – WS-Security:

Links to the WS-Security policy set bindings settings page where the WS-Security binding are configured.

Domain default bindings settings:

Use this page to specify the default policy set bindings for this security domain.

To view this administrative console page, complete the following actions:
1. Click **Security domains >** *domain_name*
2. From Web Services Bindings, click **Default policy set bindings**

Alternatively, you can view this administrative console page when multiple security domains are defined:
1. Click **Services > Policy sets > Default policy set bindings>*domain_name***
2. In the Security Domain Default Bindings collection, click *domain_name*
3. From Web Services Bindings, click **Default policy set bindings**

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Domain default provider binding:

Specifies the default provider binding for a specific security domain. You can specify `None` if you want to use the server or global default bindings for services deployed to this server. If you specify a server default binding, this binding takes precedence over the default bindings that are specified for the security domain to which the server is deployed. The name of the security domain to which each binding is scoped is displayed to the right of the name of the binding.

Note: It is a best practice to specify a binding that includes all of the policy types. This practice ensures that your default service provider binding has the necessary configuration for all policy types that you want to use.

Domain default client binding:

Specifies the default client binding for a specific security domain. You can specify `None` if you want to use the server or global default bindings for services deployed to this server. If you specify a server default binding, this binding will take precedence over the default bindings specified for the security domain to which the server is deployed. The name of the security domain to which each binding is scoped is displayed to the right of the name of the binding.

Note: It is a best practice to specify a binding that includes all of the policy types. This practice will ensure that your default client binding has the necessary configuration for all policy types that you want to use.

Modifying default bindings at the server level for policy sets

You can define default bindings for HTTP transport, JMS transport, Secure Sockets Layer (SSL) transport, WS-Addressing, WS-ReliableMessaging, and WS-Security policies.

Before you begin

To create or modify server default bindings, you must first install and configure an application server.

About this task

You can modify default cell bindings that are delivered with the product. Default bindings are not linked to a particular policy set and they provide default settings that might be used for sharing configuration information across multiple applications.

You can create default bindings for individual application servers and customize those bindings to meet your requirements. The policy set bindings are optional. If a policy set binding is not defined at the server level, then the default cell level bindings are used. To modify these default bindings, complete the following steps:

1. Open the administrative console.
2. To see which bindings are defined as the defaults for the cell or server, click **Services > Policy sets > Default policy sets bindings**.

   After seeing which bindings are defaults, you can select and edit them from the General provider or General client policy set bindings pages.
3. To edit the default bindings, see “Setting default policy set bindings” on page 846.

Results

When you finish these steps, the default policy set bindings are modified. If you created default bindings at the server level, those default bindings are used instead of the default bindings at the cell level.

What to do next

You can create application specific bindings for your policy sets. Read about creating application specific bindings for policy set attachments.

Related tasks

"Creating application specific bindings for policy set attachment" on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

"Reassigning bindings to policy sets"
After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

Related reference

"Version 6.1 default policy set bindings" on page 853
Use this page to specify Version 6.1 default policy set bindings for the cell (global security). These bindings are used for both client and provider policy set attachments within Version 6.1 applications and attachments to service applications that are deployed to a Version 6.1 server. These default bindings are used for Version 6.1 attachments unless they are overridden at the attachment point or by a Version 6.1 server default binding.

"Default policy set bindings collection" on page 850
Use this page to specify the service provider and client default bindings. The specified service provider and client bindings are used at the cell (global security) level unless these specified bindings are overridden at the attachment point, at the server, or at a security domain.

Reassigning bindings to policy sets

After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

Before you begin

To reassign a application specific binding or a default binding, there must be an assigned binding for the service artifact.

About this task

You can assign bindings to policy sets and other service artifacts. If you created a policy set and attached it to a service artifact, you can reassign the binding that was initially assigned to that policy set.

1. Open the administrative console.
2. To reassign a binding for your service provider, click **Applications > Application Types > WebSphere enterprise applications*application_name*** > **Service provider policy sets and bindings**.

   To reassign a binding for your service client, click **Applications > Application Types > WebSphere enterprise applications*application_name*** > **Service client policy sets and bindings**.
3. Select the check box for the service artifact with an attachment binding.
4. Select **Assign Binding**.
5. Select the binding that you want to assign.
6. Click **Save**, to save your changes to the master configuration.

Results

The service artifact now inherits the policy set bindings that are indicated in either the **Service provider policy sets and bindings** or **Service client policy sets and bindings** panel.

Note: Reassigning the binding for a service artifact only reassigns it for that service artifact and any inherited artifacts. It does not reassign the binding for other non-related and non-inherited attachments to the same policy set or to a different policy set.

Example

If you have the application, `app1` with an attached policy set `Username WSSecurity default`, and you want to define custom bindings, then perform the following steps:

1. Locate the application in the **Applications > Application Types > WebSphere enterprise applications*application_name*** collection.
2. Click the **app1** application.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link.
4. Select the check box for the**app1** application where the custom attachment binding is defined.
5. Click **Assign Binding** and select **New**.
6. Provide a name for the application specific bindings.
7. Add required policies to the new application specific bindings.
8. Configure the bindings.
9. Click **Save**, to save your changes to the master configuration.

What to do next

Restart the application that contains the policy set where you reassigned the bindings.

Related tasks

"Creating application specific bindings for policy set attachment" on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

Transformation of policy and binding assertions for WSDL

Web Services security does not fully support the OASIS WS-SecurityPolicy Version 1.2 standard. However, several of the policy and binding assertions supported by WebSphere Application Server can be transformed and represented as WS-SecurityPolicy Version 1.2 assertions. The supported assertions are transformed when a Web Services Description Language (WSDL) or Web Services MetadataExhange (WS-MEX) request is received in a message, and also when the client receives a policy containing WS-SecurityPolicy 1.2 assertions.

When the WebSphere Application Server receives a WSDL or WS-MEX request, some policy and binding assertions are transformed into standard assertions and included in the policy that is embedded into the WSDL. In addition, when the client receives a policy containing these WS-SecurityPolicy assertions, the assertions are transformed back into product-specific assertions so that the Application Server run time can process them. This transformation provides interoperability between Application Server and other systems that support the WS-SecurityPolicy version 1.2 standard.

The following WS-SecurityPolicy 1.2 assertions can be represented in the policy returned on WSDL, or a WS-MEX request.

EncryptSignature
: Represented in the product using XPath expressions in the encrypted elements.

EncryptBeforeSigning
: The order attribute of the encryptionInfo and signingInfo elements on the outbound section of the bindings determines the order in which the transform takes place, and whether this assertion is set. The default behavior is to sign before encrypting.

ContentEncryptedElements
: Represented using XPath expressions ending with /node() in the encrypted elements. The Application Server can consume this policy assertion, but existing XPath expressions that end with /node() are not transformed.

KerberosToken

Represented using the custom token in the policy and bindings. The Kerberos custom token assertion specifies a local name of http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ, or http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ, depending on the desired Kerberos token type. Also, there is a custom property in the bindings, com.ibm.wsspi.wssecurity.krbtoken.requireDerivedKey, which specifies use of derived keys for Kerberos. Using the local name of the custom token, along with the derived key custom property from the product representation, an equivalent Version 1.2 representation can be created.

Require*<variable>***Reference, where** *<variable>* **is one of the following: KeyIdentifier, IssuerSerial, EmbeddedToken, or Thumbprint**

Rrepresented using the type attribute on keyInfo in the bindings.

MustSupportRef*<variable>***, where** *<variable>* **is one of the following: KeyIdentifier, IssuerSerial, EmbeddedToken, or Thumbprint**

This assertion is not represented in the WebSphere Application Server policies, but the product supports all four types of references.

Protection of the SignatureConfirmation element

The SignatureConfirmation element is implicitly signed and encrypted. However, if nothing is encrypted on the response, then the SignatureConfirmation element is not encrypted, and if nothing is signed on the response, then the SignatureConfirmation element is not signed. All XPath expressions representing the signing or encryption of the SignatureConfirmation element are removed from the policy during transformation. The explicitlyProtectSignatureConfirmation attribute in the Web services security binding is provided to disable implicit signature and encryption of the SignatureConfirmation element on the response message. This provides interoperability with WebSphere Application Server Version 6.1.x. To add the attribute, check the option **Disable implicit protection for Signature Confirmation** in the Authentication and protection panel for the default policy set bindings. If the explicitlyProtectSignatureConfirmation attribute is present in the binding, all XPath expressions representing the signing or encryption of the SignatureConfirmation element remain unchanged during transformation.

SC13SecurityContextToken

Version 1.3 of the Security Context Token is supported by specifying the local name http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512, in the binding for the security context token.

RequireImpliedDerivedKeys

This is supported by adding the custom property, com.ibm.ws.wssecurity.token.generateImpliedDerivedKey, in the token generator bindings.

ExactlyOne

This assertion is transformed when callers are used. Callers specify which token to use for authentication. The ExactlyOne assertion communicates the caller tokens that the service expects. All caller options are enclosed inside the <ExactlyOne> assertion, and each option is enclosed inside the <wsp:All> assertion. As the name implies, the client sends only one of the token types. For example, in the server side bindings, using the product representation, the following caller options are specified:

```
<caller order="1">
       <jAASConfig configName="system.wss.caller"/>
       <callerIdentity uri="" localName="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#UsernameToken"/>
</caller>
<caller order="2">
       <jAASConfig configName="system.wss.caller"/>
       <callerIdentity localName="LTPA" uri="http://www.ibm.com/websphere/appserver/tokentype/5.0.2" />
</caller>
```

This assertion indicates that a UsernameToken token or an LTPA token is used as the caller. The requirement to use one of these two types of tokens is communicated to the client in the transformed policy, as in the following example:

```
<sp:ExactlyOne>
<wsp:All>
<sp:SupportingTokens>
   <wsp:Policy wsu:Id="request:token_auth">
     <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient">
       <wsp:Policy>
         <sp:WssUsernameToken10 />
       </wsp:Policy>
     </sp:UsernameToken>
 </wsp:Policy>
</sp:SupportingTokens>
</wsp:All>
<wsp:All>
<sp:SupportingTokens>
<wsp:Policy wsu:Id="request:token_auth">
     <spe:LTPAToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient" />
</wsp:Policy>
 </sp:SupportingTokens>
<wsp:All>
</sp:ExactlyOne>
```

The product-specific policy assertions LTPAToken and LTPAPropagationToken are not altered during transformation. These assertions are included in the embedded policy in the WSDL if they are present in the policy being transformed. This allows a WebSphere Application Server client and WebSphere Application Server service provider to interoperate.

Related concepts

"WSDL" on page 338
Web Services Description Language (WSDL) is an Extensible Markup Language (XML)-based description language. This language was submitted to the World-Wide Web Consortium (W3C) as the industry standard for describing Web services. The power of WSDL is derived from two main architectural principles: the ability to describe a set of business operations and the ability to separate the description into two basic units. These units are a description of the operations and the details of how the operation and the information associated with it are packaged.

Related reference

"WS-Security authentication and protection for general bindings" on page 878
Use the links on this page to configure authentication, signature, and encryption information that the policy requires when using general bindings.

Configuring the callers for general and default bindings

The caller specifies the token or message part that is used for authentication.

Before you begin

Before you can complete this task, you must create a new policy set and attach it to a service, or copy and edit one of the sample system policy sets. For more information, read the topics Creating policy sets using the administrative console and Attaching a policy set to a service artifact.

About this task

The caller is used to indicate which of the tokens on the incoming message is the caller of the request. This information is used to create authentication credentials. You can use the WebSphere Application Server administrative console to access, view and configure caller settings for tokens and message parts.

Note: WebSphere Application Server Version 7.0 provides support for multiple callers. The caller token used for authentication is the one with highest priority, based on decreasing order of preference. You can modify the order of the callers, as described in the topic Changing the order of the callers for a token or message part.

1. Create a new policy set and attach it to a service, or copy and edit a sample system policy set. Add the WS-Security policy, as described in the topic Creating policy sets using the administrative console.
2. Edit the general or default bindings for the WS-Security policy.

- To edit general provider bindings for WebSphere Application Server version 7.0, click **Services** → **Policy Sets** → **General Provider policy set bindings**. A caller is specified for the provider bindings only, not for the client bindings.
- To edit default bindings for WebSphere Application Server Version 6.x, click **Services** → **Policy Sets** → **Default policy set bindings**.

3. Navigate to the Callers panel by clicking on the **WS-Security** policy, then click the **Caller** link.
4. Click **New** to create a new caller.
5. Enter the Name and Caller identity local part information for the new caller. For more information, read about caller settings.
6. When you have finished entering the configuration information for the caller, click **Apply** to save the caller.
7. If this is the first caller created for the policy set, the caller is automatically assigned as the highest priority caller, with an order of 1 (one). If other callers are already defined, the new caller is added at the end of the ordered list and is automatically assigned the lowest priority. You can change the order of the callers using the Move up and Move down buttons.

Results

When assigning orders to callers for migrated bindings, the callers are initially displayed with no order attribute. You cannot save the bindings until you assign order attributes to all the callers. Use the **Move up** and **Move down** buttons to change the order of the callers until they are in the correct order.

Signing and encrypting message parts using policy sets

With Web services, you can sign message parts, encrypt message parts, or both, based on the quality of service defined for a policy set. You can accomplish these actions by defining the binding information in a custom attachment binding.

Before you begin

Before you begin this task, attach a policy set to a service artifact such as an application, service or endpoint and create a custom attachment binding. Read about creating custom attachment bindings for policy sets. The policy set that is attached to the service artifact must include a WS-Security policy that specifies message parts to be signed or encrypted. Read about securing message parts using the administrative console.

About this task

To sign message parts, encrypt message parts, or both, based on the quality of service defined for a policy set, perform the following steps:

1. Open the administrative console.
2. To sign and encrypt message parts for a service provider, click **Applications > Enterprise applications** > ***application_name*** > **Service provider policy sets and bindings**. To sign and encrypt message parts for a service client, click **Applications > Enterprise applications** > ***application_name*** > **Service client policy sets and bindings**.
3. Click the binding name link of the service artifact with a custom attachment binding.
4. If the binding does not contain WS-Security policy set bindings, then click **Add** and select **WS-Security** from the list.
5. Click **WS-Security** policy set bindings.
6. Click **Authentication and protection**. The resulting panel contains the following four tables:
   - Protection tokens: Specifies the tokens that are defined for the symmetric or asymmetric signature and encryption policies in the policy set.
   - Authentication tokens: Specifies the tokens that are defined for the request and response token policies.

- Request message signature and encryption protection: Specifies the message parts that are defined in the Request message part protection for the policy set.
- Response message signature and encryption protection: Specifies the message parts that are defined in the response message part protection in the policy set.

Initially, each table displays information that is generated based on the policy set which is attached to the service artifact. The possible configuration objects based on the policy set are displayed. The Status column indicates whether the object is currently configured in the custom attachment binding.

7. If the protection tokens have a status of **Not configured**, then create the protection tokens by clicking the default name, verifying the default values. Click **OK**.
8. [Optional] If you use the X.509 protection tokens, then you must configure the keystores and keys to be used to sign, verify, encrypt or decrypt message parts. You might need to also configure keystores and keys when using custom protection tokens, depending on the requirements of the custom tokens. When using a security context token for protection (secure conversation), you do not need to configure keystores or keys. If you need to configure the keystores and keys, then perform the following actions:
   a. Click the token name link.
   b. Click the **Callback handler** link under Additional bindings. If the Callback handler link is not click-able, click **Apply**, then click the **Callback handler** link.
   c. Either use a predefined keystore or custom keystore. To use a predefined keystore, select the keystore from the list. To use a custom keystore, select **Custom** from the list and click the **Custom key store configuration** link to specify the configuration.
   d. Click **OK**.
9. Click the name of the request or response message part reference to be signed or encrypted. The Protection column displays whether the message part is signed or encrypted based on the policy set.
10. Specify a name for the message part.
11. For encrypted parts, select the type of encryption from **Usage of key information references**. For asymmetric encryption, or X.509, select **Key encryption**. For symmetric encryption, or secure conversation, select **Data encryption**.
12. [Optional] For encrypted parts, select the **Include time stamp** or **Include nonce** options to include a time stamp or nonce in the encrypted message part. You can include one or both of these options in the encrypted message part.
13. For signed parts, specify one or more Message part references. Select a reference from the Available column and click **Add**.
14. [Optional] For signed parts, you can also choose to add a time stamp or nonce to the signed message part. Select a Message part reference from the Assigned column and click **Edit**. Select the **Include time stamp** or the **Include nonce** options to include a time stamp or nonce in the signed message part. You can select one or both of these options in the signed message part.
15. If there are no available key information entries, then create one using the following actions:
   a. Click **New**.
   b. Specify a name.
   c. Select a protection token from the Token generator or Consumer name list.
   d. Click **OK**.
16. Select a key information entry from the Available list and click **Add**.
17. [Optional] Specify custom properties if needed.
   a. To use Message Transmission Optimization Mechanism (MTOM) for the cipher text of the encrypted data, add the custom property, com.ibm.wsspi.wssecurity.enc.MTOM.Optimize, with value `true` to outbound encrypted parts for client requests or server responses.
   b. To use encryption headers as described in the WS-Security 1.0 specification instead of the encrypted header support described in WS-Security 1.1, add the custom property,

com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.0, with value `true` to outbound encrypted parts for client requests or server responses.

For Web Services Security Version 1.1 behavior that is equivalent to WebSphere Application Server versions prior to version 7.0, specify the com.ibm.wsspi.wssecurity.encryptedHeader.generate.WSS1.1.pre.V7 property with a value of true on the <encryptionInfo> element in the binding. When this property is specified, the <EncryptedHeader> element includes a wsu:Id parameter and the <EncryptedData> element omits the Id parameter. This property should only be used if compliance with Basic Security Profile 1.1 is not required.

18. Click **OK**.
19. Click **Save**, to save the changes to the master configuration.

Results

When you finish this task, the message parts are signed and encrypted, or both, based on the configuration used when communicating with the service artifact.

Example

You have an application, `app1`, with an attached policy set, RAMP default and a custom attachment binding, `myBinding`, and you want to sign and encrypt the message parts.

1. Click the `app1` application in the **Applications** > **Enterprise Applications** collection.
2. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link.
3. Click the `myBinding` link.
4. [Optional] If WS-Security is not listed, then select **Add** > **WS-Security**.
5. Click the **WS-Security** link.
6. Click the **Authentication and protection** link.
7. In the Protection tokens table, click each of the four links and **OK** on the resulting panel. Each entry is now shown as **Configured** in the Status column.
8. In the Request message signature and encryption protection table, click **request:app_encparts**. Specify the name, `requestEncParts`.
9. Click **New** from Key information. Specify the name, `requestEncKeyInfo`.
10. Select **SymmetricBindingRecipientEncryptionToken**, and click **OK**.
11. Select **requestEncKeyinfo** in the Available list, and click **Add**. Click **OK**.
12. In the Request message signature and encryption protection table, click **request:app_signparts**.
13. Specify the name, `requestSignParts`.
14. Click **New** from Key information. Specify a name of `requestSignKeyInfo`.
15. Select **SymmetricBindingInitiatorSignatureToken**, and click **OK**.
16. Select **requestSignKeyinfo** in the Available list, and click **Add**. Click **OK**.
17. Repeat steps 8 to 16 for the links in the Response message signature and encryption protection table.
18. Click **Save**, to save the changes to the master configuration.

What to do next

Start the application.

Related tasks

“Attaching a policy set to a service artifact” on page 901
Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

“Creating application specific bindings for policy set attachment” on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

“Configuring the WS-Security policy” on page 933
When working with policy sets in the administrative console, you can customize policies to ensure message security. The WS-Security policy can be configured to apply a message security (WS-Security) profile to requests. Message security policies are applied to requests and enforced on responses to support interoperability.

“Securing message parts using the administrative console” on page 957
If you are working with policy sets, then you can secure message parts using the administrative console. To secure message parts with WS-Security using policy sets, you must define the elements for the message parts to be protected in the WS-Security policy within a policy set.

Signed or Encrypted message part settings:

Use this page to configure or create new signed or encrypted message parts. Message part bindings define how the part (which is defined in a policy set) is handled.

You can configure or create new signed or encrypted message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Authentication and protection** link in the Main message security policy bindings section.
4. Select a signature or an encrypted message part in the Request message signature and encryption protection section or the Response message signature and encryption protection section.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.
7. Select a signature or an encrypted message part in the Request message signature and encryption protection section or the Response message signature and encryption protection section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the name of the message part reference. The name field displays the name of the part reference you are editing, or you can enter a name if you are creating a message part reference.

Include time stamp:

This check box is available on this panel if you are configuring encryption protection and it specifies whether to include a time stamp. Select this check box to indicate that a time stamp is included or leave it unchecked to indicate that the time stamp is not included with the part reference.

For default bindings, to specify if a time stamp is included for signature protection, click the Signed part reference default link under the Additional bindings section.

For application specific bindings, to specify if a time stamp is included for signature protection, highlight an assigned signature message part reference and click **Edit**. The time stamp check box is located in the Reference section.

Include nonce:

This check box is available on this panel if you are configuring encryption protection and it specifies whether to include nonce. Select this check box to indicate that a nonce is to be used or leave it unchecked to indicate that nonce is not to be included with this part reference.

For default bindings, to specify if a nonce is included for signature protection, click the Signed part reference default link under the Additional bindings section.

For application specific bindings, to specify if a nonce is included for signature protection, highlight an assigned signature message part reference and click **Edit**. The nonce check box is located in the Reference section.

Usage of key information reference:

This field is available on this panel if you are configuring encryption protection and it specifies that the encryption key information is either data encryption key information or key encryption key information. Select **Data encryption** for symmetric algorithms and **Key encryption** for asymmetric algorithms.

Click one of the following radio buttons:

Data encryption
Indicates that the key information is used for data encryption.

Key encryption
Indicates that the key information is key encryption key information.

Key information (Request):

If you are configuring a request message signature or encryption protection, this field specifies the key information for a token request message part. This section provides interactive fields to assign the key information.

The **Available** field contains a listing of available key information entries for the message part. The **Assigned** field contains a listing of one or more of the key information entries that are assigned to the message part. Use the following actions to work with multiple request message part key information entries:

Button	Resulting action
Add >	Add the selected key information entry in the **Available** list to the **Assigned** list.
New...	Create a new key information entry.
< Remove	Remove the selected key information entry from the **Assigned** list.

Key information (Response):

If you are configuring a response message signature or a response encryption protection, this field specifies the key information for a token response message part. This field provides a menu used to assign the key information. You can only assign one key information entry for response message parts. The **New** button enables you to add a new key information entry to the menu for selection.

Custom properties – Name:

Specifies the name of the custom property to be used.

Custom properties are not initially displayed in this column. The following actions are available:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Edit	Specifies that you can edit the selected custom property. Select this action to provide input fields and create the listing of cell values for editing. The Edit button is not available until at least one custom property has been added.
Delete	Removes the selected custom property.

Custom properties – Value:

Specifies the value of the custom property to be used. With the Value entry field, you can edit, enter or delete the value for a custom property.

Additional bindings – Signed part reference default:

If you are configuring signature protection, this section is displayed on this panel. It links to a panel where you can configure part reference properties such as including a time stamp or nonce and transform algorithms. Part reference properties include the transform algorithms used to protect the message part.

Changing the order of the callers for a token or message part

Specifying a caller in default and general bindings indicates which token or tokens to use to create authentication credentials. When there are multiple tokens on an incoming message, the order of the callers determines which token is used for the credentials. You can rearrange the order of the callers using the administrative console.

Before you begin

Before you can complete this task, you must create a new policy set and attach it to a service, or copy and edit one of the sample system policy sets. For more information, read the topics Creating policy sets using the administrative console and Attaching a policy set to a service artifact. You can also create multiple new callers in the default provider bindings associated with the policy set, as described in the

topic Configuring the callers for general and default bindings

1. Edit the bindings for the policy by clicking **Services** → **Policy sets** → **General Provider policy set bindings**, then click on the name of the bindings. A caller is specified for the provider bindings only, not for the client bindings.
2. Navigate to the Callers panel by clicking on the policy name, then click the **Caller** link.
3. In the caller collection table, the callers are listed with the order of each caller displayed in the **Order** column. The order number indicates the order of preference in which the callers are used when multiple authentication tokens are received on an incoming message. Use the **Move up** and **Move down** buttons to change the order of the callers.
   a. To change the order of a caller to a higher priority, click the selection box next to the caller name, then click the **Move up** button. When a caller is moved up in priority, a caller that is above it will be moved down.
   b. To change the order of a caller to a lower priority, click the selection box next to the caller name, then click the **Move down** button.

Example

The order attribute is assigned only for callers on bindings in WebSphere Application Server Version 7.0. Bindings created with earlier versions of WebSphere Application server may have callers, but these callers do not have an order attribute. These callers can appear in the Callers collection table, but do not have an order number in the order column. If these bindings were migrated to Version 7.0, then order attributes must be assigned before saving and using these bindings. You can use the **Move up** and **Move down** buttons to assign orders to the callers.

Policy set bindings settings

Use this page to view or define general or application specific bindings configuration information that is specific to a system for policies that you can associate with the selected policy set. Use the links on this page to work with bindings for each specific policy.

To view this administrative console page when you are editing a general binding, click **Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**.

To view this administrative console page when you are creating or editing an application specific binding, complete the following actions:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click **Service provider policy sets and bindings** or **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding by clicking the binding name in the table if the binding has been assigned to an attachment. You must have previously attached a policy set and assigned an application specific binding.
5. [Optional] To edit a default cell binding or default server binding, click either**Services > Policy sets > General provider policy set bindings** or **Services > Policy sets > General client policy set bindings**.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

About Policy set bindings

Policy set bindings contain platform-specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Each policy set attachment to a service provider or service client must have exactly one binding. When you create a policy set attachment, the

general default bindings are used initially. When general bindings are used in association with a policy set attachment, the cell-level general bindings are applied at run time. If application server level bindings exist, the server-level general bindings override the cell-level definition. General bindings specify configuration for both service client and service provider attachments and the general bindings are not tailored to a specific policy set or application. When you define server-level general bindings, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP Transport, that you want to override the general binding and you must fully configure the bindings for each policy that you have added.

An application specific binding is a named binding that you create. Application specific bindings enable you to provide platform-specific configuration information for specific policy set attachments. When you create an application specific binding, the available binding configuration options are tailored to the definitions in the attached policy set. You can reuse application specific bindings for multiple service resources within an application. For example, if you create a trust service specific binding, that binding can be reused only for trust service attachments. When you create an application specific binding for a policy set attachment, the binding begins in a completely unconfigured state. You must add each policy, such as WS-Security or HTTP Transport, that you want to override the general binding and you must fully configure the bindings for each policy that you have added.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Bindings configuration name:

Specifies the name of the policy set bindings configuration. The binding name is not editable when you are editing a binding. When you are creating a new binding, you must specify the binding name.

Note: If you are running a Version 6.1 application, the Binding configuration name is displayed as Version 6.1 default policy set bindings.

Use the following actions to create, edit, or delete policy set bindings.

Button	Resulting action
Add	Adds the selected policy set binding to the application.
Delete	Removes the selected policy set binding from the application.

Policies – HTTP transport:

Links to the HTTP transport policy configuration settings page where you define the HTTP transport settings. The HTTP features and HTTP connection polices are applied to outbound messages. The response listener policy is enforced on inbound messages.

Policies – SSL transport:

Links to the SSL transport policy configuration settings page where you define the SSL transport settings.

Policies – JMS transport:

Links to the JMS transport policy configuration settings page where you define the JMS transport settings.

Policies – WS-Addressing:

Links to the configuration settings page for the WS-Addressing policy. In a Network Deployment environment, use this page enable or disable workload management. Otherwise, you can attach the WS-Addressing policy set to service resources. No additional configuration is required.

Policies – WS-ReliableMessaging:

Links to the panel where the WS-ReliableMessaging bindings are configured.

Policies – WS-Security:

Links to the WS-Security policy set bindings settings page where the WS-Security bindings are configured.

Policy set bindings settings for WS-Security

Use this page to view, define or configure general bindings and application specific properties for the WS-Security policy. You can configure the main policy or the secure conversation bootstrap policy by editing the general bindings.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

To use this administrative console page to view the general default bindings, click **Services > Policy sets > Default policy set bindings**. You can use this navigation path for viewing only. To edit or configure the general default bindings, complete the following steps:

1. Navigate to the general bindings collection panel by clicking**Services > Policy sets > General client policy set bindings** or **Services > Policy sets > General provider policy set bindings** path.
2. Click a general binding in the Name column.
3. Click the **WS-Security** policy in the Policies table.

If you choose to use a sample binding that is provided in the product, you must edit the sample user name and password that are provided for the Username token and LTPA token. The values provided are only examples; to use them successfully, you must modify the values for your own environment. You can change the user ID and password for authentication using a scripting command or by editing a copy of the general binding.

The following configuration links are provided for both the main security policy and for secure conversation bootstrap policy bindings.

Authentication and protection:

Links to the collection of policy authentication and protection configuration settings. Click this link to access the collection of authentication and protection settings where you can configure authentication, signature, and encryption information that the policy requires.

Keys and certificates:

Links to the collection of WS-Security policy keys and certificates.

Caller:

Links to a panel to configure the caller settings. The caller specifies the token or message part that represents the identity to be set in the caller subject of the service.

The caller settings are available only for the service provider policy sets and bindings. The caller settings are not available for service client policy sets and bindings.

Message expiration:

Links to a panel to define settings for message expiration. When you enable message expiration, the message expires after the specified interval.

Custom properties:

Links to a panel where you can specify custom properties that apply to both inbound and outbound messages or specify properties that apply only to inbound or only to outbound messages.

Custom property settings:

Use this page to configure name-value pairs for custom binding properties, where the name is a property key and the value is a string value that can be used to set internal system configuration properties. You can specify custom properties that apply to both inbound and outbound messages, or properties that apply only to inbound messages, or only to outbound messages.

To view this administrative console page, click **Applications** → **Application types** → **Websphere enterprise applications** → ***application name*** → **Service provider policy sets and bindings** → ***binding name*** → **WS-Security** → **Custom properties**.

Click **New** to create a new custom property, or click the check box for an existing property name. Click **Delete** to delete an existing property.

Name:

Specifies the name, or key, for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was.` because this prefix is reserved for properties that are predefined in the product.

Value:

Specifies the value paired with the specified name.

Keys and certificates

Use this page to link to key and certificate binding configuration panels. This panel defines key and certificate bindings for JAX-WS Web services only. These keys and certificates can be centrally managed by the product or in an external keystore.

You can define key and certificate bindings for message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > General provider policy set bindings** (for provider bindings), or **Services > Policy sets > General client policy set bindings** (for client bindings).
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Keys and certificates** link in the Main message security policy bindings section.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.

2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Keys and certificates** link in the Main message security policy bindings section.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Key information – Name:

Specifies the key information name. The key names listed in this field are links that are used to define key information attributes. Key information attributes define how cryptographic keys are generated or consumed.

Use the following buttons to work with this table:

Button	Resulting Action
New Inbound...	Creates a new inbound key information name.
New Outbound...	Creates a new outbound key information name.
Delete	Removes the selected key information name listing.

Key information – Type:

Specifies the type of key information.

Key information – Direction:

Specifies the whether the direction of the key is inbound or outbound. .

Certificate store – Name:

Specifies the certificate store name. The certificate store names listed in this table are used to configure certificate stores.

Use the following actions to work with this table:

Button	Resulting Action
New Inbound...	Creates a new inbound certificate store.
New Outbound...	Creates a new outbound certificate store.
Delete	Removes the selected certificate store.

Certificate store – Direction:

Specifies whether the direction of the certificate store is inbound or outbound.

Trust anchor – Name:

Specifies the trust anchor name. The trust anchor names in this table are links that are used to configure trust anchor certificate stores.

You can use the following buttons to work with this table:

Button	Resulting Action
New...	Creates a new trust anchor entry.
Delete	Removes the selected trust anchor.

Trust anchor – Keystore:

Specifies the type of keystore for the trust anchor.

Key information settings:

Use this page to configure the key information for the selected policy set binding. Key information attributes define how cryptographic keys are generated or consumed.

You can configure the key information for the selected policy set binding when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > General provider policy set bindings** or **General client policy set bindings**.
2. Click on a binding name in the **Name** column.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Keys and certificates** link in the Main message security policy bindings section.
5. Click a key in the **Name** column of the Key information table.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Keys and certificates** link in the Main message security policy bindings section.
7. Click a key in the **Name** column of the Key information table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the unique name for the key information configuration.

The key information name field displays the unique name of the key that is being configured if you are editing a key. If you are creating one, enter a unique name.

Type:

Lists the type of key reference.

This field appears only if you selected an encryption or signing key for the generator binding, such as gen_signkeyinfo, gen_signsctkeyinfo, gen_encsctkeyinfo, or gen_enckeyinfo.

You can select one of the following key types from this list:

Key identifier
The associated attribute in the binding file is KEYID.

Security token reference
The associated attribute in the binding file is STRREF.

Embedded token
The associated attribute in the binding file is EMB.

X.509 issuer name and issuer serial
The associated attribute in the binding file is X509ISSUER.

Thumbprint
The associated attribute in the binding file is THUMBPRINT.

The Thumbprint key information type requires a keystore with the public and private key pair instead of a shared key.

Data type: Selection list

Token generator or consumer name:

Specifies the name of the token generator or consumer. Specifies a unique name for the token configuration.

The token generator or consumer name field displays the name of the pre-configured tokens that can be used in the key information configuration if you are editing a key or creating a new key.

You can select a token generator or consumer name from this list. The list of names changes, depending on whether the key information selected is for inbound (consumer) keys or outbound (generator) keys. For keys with outbound direction, the list of defined token generators is displayed. For keys with inbound direction, the list of defined token consumers is displayed.

Data type: String

Direction:

Specifies whether the direction of the key is inbound or outbound.

The direction of generator tokens are outbound whereas the direction for consumer tokens and decryption keys are inbound.

Data type: String
Default values: Inbound (for consumer bindings) or Outbound (for generator bindings)

Requires derived keys:

Specifies whether the key information requires derived keys.

Explicit derived keys
Requires that derived keys be explicitly specified with a WS-SecureConversation <DerivedKeyToken> element.

Implicit derived keys
Requires that derived keys be implicitly specified with a WS-SecureConversation Nonce attribute on the WS-Security <SecurityTokenReference> element.

Override Defaults:

Specifying derived key values overrides the derived key information that the runtime generates by default.

Note: It is recommended that you do not override the following optional attributes. Web Services Security automatically provides default values for each attribute. Overriding the default values might be required if the service is running cross-vendors. The vendors can use different attribute values for derived key generation.

Key length
Specifies the derived key length. If an override value is not specified, the default value is provided based on the algorithm suite policy assertion. It is recommended that you leave this field empty so the default value can be used. Valid values for the key length range between 16 and 32.

Nonce length
Specifies the nonce length. A nonce is generated for each request, and included for derived key generation. This value is optional, and if an override value is not specified, a default value is used to generate the nonce. A valid value for the nonce length is any integer between 16 and 128.

Client label
Specifies the client label. The label is used in the P_SHA-1 function to generate the derived key. If unspecified, the default value used is `WS-SecureConversation`.

Service label
Specifies the service label. The label is used in the P_SHA-1 function to generate the derived key. If unspecified, the default value used is `WS-SecureConversation`.

Custom properties:

Specifies additional configuration settings that token types might require.

Custom properties are arbitrary name-value pairs of data.

This table lists custom properties. Use custom properties to set internal system configuration properties. You are not required to define a custom property when you define a custom token.

Select:

Specifies custom properties that you can add, edit, or delete from policy set bindings.

Click **New** to add and define a new custom property.

For existing custom properties, select the check box for the name of the custom property, and click one of the following actions:

Action	Description
New	Creates a new custom property entry. To add a custom property, enter the name and value.

Action	Description
Edit	Specifies that you can edit the selected custom property. Click this option to provide input fields and create the list of cell values to edit. At least one custom property must exist before the Edit option is displayed.
Delete	Removes the selected custom property.

Data type: Check box (unchecked)

Name:

Specifies the name of the custom property that you can use with default policy set bindings.

Custom properties are arbitrary name-value pairs of data. Custom properties are not initially displayed in this column until at least one custom property has been added.

Data type: String

Value:

Specifies the custom property value.

This column displays the value for the custom property (for example, `true`). The value can be a string or the value can be a true or false Boolean value.

Data type: String or Boolean

Certificate store settings:

Use this page to specify the location where certificates are stored. You can reference certificate revocation for service generators or consumers.

You can specify the location where certificates are stored when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Keys and certificates** link in the Main message security policy bindings section.
4. Click the ***certificate_store_name*** link in the Certificate store section.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.

5. Click the **WS-Security** policy in the Policies table.
6. Click the **Keys and certificates** link in the Main message security policy bindings section.
7. Click the ***certificate_store_name*** link in the Certificate store section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the name of the certificate store. The name field displays the name of the certificate store if you are editing a certificate store, or enter a name if you are creating a new certificate store.

Revoked certificates – Full path:

Specifies, in the Revoked certificates table, the paths for any certificates that are revoked. The Full Path column of this table lists any certificates that have been revoked.

You can add, edit, or remove these entries with the following buttons:

Button	Resulting Action
New	Creates a revoked generator or consumer certificate store.
Delete	Removes the selected revoked generator or consumer certificate store.
Edit	Allows you to edit the applied entries selected in the checkbox. This button is only displayed if revoked certificates exist in your configuration.

Intermediate X.509 certificates – Full Path:

Specifies, for the consumer certificate store only, the paths for any intermediate X.509 certificates. The Full Path column of this table lists any intermediate X.509 certificate stores. This table is only displayed for the consumer version of this panel. It is not valid for generator certificate stores.

Note: If the certificate store is outbound, the Intermediate X.509 certificates field is not displayed.

You can create, edit, or remove intermediate X.509 certificates with the following buttons:

Button	Resulting Action
New	Creates an intermediate X.509 consumer certificate store.
Delete	Removes an intermediate X.509 consumer certificate store.
Edit	Allows you to edit the applied entries selected in the checkbox.

Trust anchor settings:

Use this page to specify the trust anchor configuration. These trust anchor certificates are used to validate the X.509 certificate that is embedded in the SOAP message.

Use this information to configure a trust anchor. Trust anchors point to keystores that contain trusted root or self-signed certificates. This information enables you to specify a name for the trust anchor and the information that is needed to access a keystore. The application binding uses this name to reference a predefined trust anchor definition in the binding file (or the default).

You can configure a trust anchor when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Keys and certificates** link in the Main message security policy bindings section.
4. Click a name link in the **Name** column of the Trust anchor table.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Keys and certificates** link in the Main message security policy bindings section.
7. Click a name link in the **Name** column of the Trust anchor table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the unique name that is used by the application binding to reference a predefined trust anchor definition in the default binding.

A trust anchor specifies the keystore that contains trusted root certificates. This field displays the name for the trust anchor that is being edited. If you are creating a new trust anchor configuration, enter a unique name.

Keystore files contain public and private keys, root certificate authority (CA) certificates, the intermediate CA certificate, and so on. Keys that are retrieved from the keystore files are used to sign and validate or encrypt and decrypt messages or message parts.

Data type: String

Centrally managed keystore:

Specifies to use a centrally managed keystore. After selecting the **Centrally managed keystore** option, choose one of the centrally managed keystore names from the list. Centrally managed keystores can be managed in the administrative console by clicking these links: **Security > SSL certificate and key management > Key stores and certificates**.

Click the radio button to enable the **Name** field. Select a keystore from the list.

Data type: Radio button
Default value: Unselected

External keystore:

Specifies a keystore using a keystore path, keystore type and keystore password. The keystore file format is determined by the keystore type. The default trust anchor in the default binding uses an external keystore.

Select the radio button to enable an external keystore.

Data type: Radio button
Default value: Selected

Full path
Specifies the full path to the location of the keystore.

If the keystore is file-based, the location can reference any path in the file system of the node where the trust anchor keystore is located. The trust anchor defined in the default bindings is:

```
${USER_INSTALL_ROOT}/etc/ws-security/samples/dsig-receiver.ks
```

Note: Do not use the sample keystore files in a production environment. These samples are provided for testing purposes only.

Data type: String

Type Specifies the type of keystore when the external keystore is enabled.

The type specifies the implementation for keystore management. Click a keystore type from the list provided. The selection list is returned by java.security.Security.getAlgorithms(″KeyStore″).

The IBM Java Cryptography Extension (IBMJCE) supports the following file-based keystore types: JKS, JCEKS,PKCS12, and CMSKS.

- Use the JKS option if you are not using Java Cryptography Extensions (JCE).
- Use the JCEKS option if you are using Java Cryptography Extensions.
- Use the PKCS12 option if your keystore uses the PKCS#12 file format.
  - A key.p12 file or a trust.p12 file are examples of PKCS12 type keystores.
- Use the CMSKS option if your keystore uses the Certificate Management Services (CMS) format.

Password
Specifies the password that is needed to access the keystore file.

Use the password to protect the keystore. The password is used to access the named keystore and the password is also the default password that is used to store keys within the keystore.

The default trust anchor in default binding uses an external keystore. The password for the external keystore is: `server`. It is recommended that you change the default password as soon as possible.

Data type: String
Default value: WebAS or cell name

Confirm password
Confirms the password entered in the Password field.

Enter the password that is used to open the keystore file or device again. By entering the same password that was entered in the Password field again, you confirm the password.

Data type: String

WS-Security authentication and protection

Use the links on this page to configure authentication, signature, and encryption information that the policy requires.

You can configure authentication, signature, and encryption information for tokens and message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a general provider policy set binding or general client policy set binding, complete the following actions:

1. Click **Services > Policy sets > General provider policy set bindings or General client policy set bindings**.
2. Click **Provider sample** or **Client sample**.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Authentication and protection** link in the Main Message Security Policy Bindings section.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned an application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

WS-Security authentication and protection for general bindings:

Use the links on this page to configure authentication, signature, and encryption information that the policy requires when using general bindings.

You can configure authentication, signature, and encryption information for tokens and message parts when you are editing a general binding.

To view this administrative console page when you are editing a general binding at the cell level, complete the following actions:

1. Click **Services > Policy sets > General provider policy set bindings** or **General client policy set bindings**.
2. Click on the name of the bindings you want to edit.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Authentication and protection** link in the Main message security policy bindings section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Disable implicit protection for Signature Confirmation:

Specifies whether implicit protection of the SignatureConfirmation element is enabled or disabled.

The explicitlyProtectSignatureConfirmation attribute in the Web services security binding is provided to disable implicit signature and encryption of the SignatureConfirmation element on the response message. If this checkbox is selected, the attribute is added and implicit protection is disabled. This provides interoperability with earlier versions of WebSphere Application Server.

Default: Not selected (implicit protection is enabled)

Protection tokens – Protection token name:

Specifies a list of protection tokens that can be configured in the Protection tokens table.

The following actions are available for general bindings:

Button	Resulting Action
New Token	Creates a new protection token type.
Delete	Removes the selected protection token type.

Protection tokens – Usage:

Specifies the policy assertion usage names that you can customize in the Protection tokens table.

For the usage field, the following options are available for the general bindings:

- Asymmetric encryption generator
- Asymmetric encryption consumer
- Asymmetric signature generator
- Asymmetric signature consumer
- Symmetric encryption generator
- Symmetric encryption consumer
- Symmetric signature generator
- Symmetric signature consumer

Authentication tokens – Authentication token name:

Specifies a list of authentication tokens that you can customize in the Authentication tokens table when using general bindings.

If you are working with a Username token or LTPA token that is using general bindings, the user names and passwords might have been provided as examples. When you click a Username token or LTPA token link, you need to update the values for these token types using the Callback handler link found on the Authentication token settings page.

The following actions are available for general bindings:

Button	Resulting Action
New Token	Creates a new authentication token type.
Delete	Removes the selected authentication token type.

Authentication tokens – Usage:

Specifies the usage names from the Authentication tokens table for general bindings.

The following options are available for general bindings:
- Inbound
- Outbound

Request message signature and encryption protection – Name:

Specifies a unique name to identify the request message part from the Request message signature and encryption protection table that is protected.

The following actions are available for general bindings. The Move up and Move down actions are available only when using service client policy sets and bindings.

Button	Resulting Action
New Signature	Creates a new signature.
New Encryption	Creates a new encryption protection.
Delete	Removes the selected request message part.
Move up	Moves the selected request message part up in the order.
Move down	Moves the selected request message part down in the order.

Request message signature and encryption protection – Protection:

Specifies the type of protection from the Request message signature and encryption protection table. This field displays the type of protection enabled for the general binding.

Response message signature and encryption protection – Name:

Specifies a unique name to identify the response message part from the Response message signature and encryption protection table that is protected.

The following actions are available for general bindings. The Move up and Move down actions are available only when using service provider policy sets and bindings.

Button	Resulting Action
New Signature	Creates a new response message signature.
New Encryption	Creates a new encryption.
Delete	Removes the selected response message part.
Move up	Moves the selected response message part up in the order.
Move down	Moves the selected response message part down in the order.

Response message signature and encryption protection – Protection:

Specifies the type of protection enabled from the Response message signature and encryption protection table. This field displays the type of protection enabled for the response message part.

Response message signature and encryption protection – Order:

Specifies the order in which the signatures and encryptions occur. Use the **Move up** and **Move down** actions to order the list of protection types in this table.

WS-Security authentication and protection for application specific bindings:

Use the links on this page to configure authentication, signature, and encryption information that the policy requires when using application specific bindings.

You can configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Disable implicit protection for Signature Confirmation:

Specifies whether implicit protection of the SignatureConfirmation element is enabled or disabled.

The explicitlyProtectSignatureConfirmation attribute in the Web services security binding is provided to disable implicit signature and encryption of the SignatureConfirmation element on the response message. If this checkbox is selected, the attribute is added and implicit protection is disabled. This provides interoperability with earlier versions of WebSphere Application Server.

Default: Not selected (implicit protection is enabled)

Protection tokens – Protection token name:

Specifies a list of protection tokens that can be configured in the Protection tokens table for application specific bindings.

The following actions are available for application specific bindings:

Button	Resulting Action
Unconfigure	Removes the selected protection token from the binding.

Protection tokens – Protection token type:

Specifies the protection token type for application specific bindings.

Protection tokens – Usage:

Specifies the policy assertion usage names that you can customize in the Protection tokens table.

For the usage field, the following options are available for the application specific bindings:

- Asymmetric encryption generator
- Asymmetric encryption consumer
- Asymmetric signature generator

- Asymmetric signature consumer
- Symmetric encryption generator
- Symmetric encryption consumer
- Symmetric signature generator
- Symmetric signature consumer

Protection tokens – Status:

Specifies the status of the protection token when using application specific bindings. The valid values are configured, not configured, or incompatible.

Authentication tokens – Security token reference:

Specifies a list of authentication tokens that you can customize in the Authentication tokens table when using application specific bindings.

The following actions are available for application specific bindings:

Button	Resulting Action
Unconfigure	Removes the selected authentication token from the binding.

Authentication tokens – Authentication token type:

Specifies the authentication token type for the security token reference when using application specific bindings.

Authentication tokens – Usage:

Specifies the usage names from the Authentication tokens table for application specific bindings.

The following options are available for application specific bindings:
- Inbound request
- Outbound request
- Inbound response
- Outbound response

Authentication tokens – Status:

Specifies the status of the authentication token form the Authentication tokens table for application specific bindings. The valid values are configured, not configured, or incompatible.

Request message signature and encryption protection – Request message part reference:

Specifies the name of the request message part in the policy from the Request message signature and encryption protection table that is protected.

The following actions are available for application specific bindings. The **Move up** and **Move down** actions are available only when using Service client policy sets and bindings.

Button	Resulting Action
Unconfigure	Removes the selected request message part from the binding.
Move up	Moves the selected request message part up in the order.

Button	Resulting Action
Move down	Moves the selected request message part down in the order.

Request message signature and encryption protection – Protection:

Specifies the type of protection from the Request message signature and encryption protection table. This field displays the type of protection enabled for the application specific binding.

Request message signature and encryption protection – Order:

Specifies the order in which signatures and encryptions occur when using service client policy sets and bindings. Use the **Move up** and **Move down** actions to order the list of protection types in this table.

Request message signature and encryption protection – Status:

Specifies the status of the request message signature and encryption protection token when using application specific bindings. The valid values are configured, not configured, or incompatible.

Response message signature and encryption protection – Response message part reference:

Specifies the name of the response message part in the policy from the Response message signature and encryption protection table that is protected.

The following actions are available for application specific bindings. The **Move up** and **Move down** actions are available only when using Service provider policy sets and bindings.

Button	Resulting Action
Unconfigure	Removes the selected response message part from the binding.
Move up	Moves the selected response message part up in the order.
Move down	Moves the selected response message part down in the order.

Response message signature and encryption protection – Protection:

Specifies the type of protection enabled from the Response message signature and encryption protection table. This field displays the type of protection enabled for the response message part.

Response message signature and encryption protection – Order:

Specifies the order in which signatures and encryptions occur when using service provider policy sets and bindings. Use the **Move up** and **Move down** actions to order the list of protection types in this table.

Response message signature and encryption protection – Status:

Specifies the status of the response message signature and encryption protection token when using application specific bindings. The valid values are configured, not configured, or incompatible.

Protection token settings (generator or consumer):

Use this page to configure protection tokens. Protection tokens sign messages to protect integrity or encrypt messages to provide confidentiality.

You can add protection token settings for message parts when you are editing general provider or client policy set bindings. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a general provider binding, complete the following actions:

1. Click **Services** → **Policy sets** → **General provider policy set bindings**.
2. Click on the name of the binding you want to edit.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Authentication and protection** link in the security policy bindings section.
5. Click **New token** to create a new token generator or consumer, or click an existing consumer or generator token link from the Protection Tokens table.
6.

To view this administrative console page when you are editing a general client binding, complete the following actions:

1. Click **Services** → **Policy sets** → **General client policy set bindings**.
2. Click on the name of the binding you want to edit.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Authentication and protection** link in the Main message security policy bindings section.
5. Click **New token** to create a new token generator or consumer or click an existing consumer or generator token link from the Protection Tokens table.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications > Websphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the security policy bindings section.
7. Click a consumer or generator token link from the Protection Tokens table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the token generator or consumer name. Enter a name in this field when you create a new token.

Token type:

Specifies the type of token. When using bindings, the token type is determined from the policy and cannot be edited.

Valid values are:

- LPTA Token V2.0
- Secure Conversation Token V1.3
- Secure Conversation Token V200502
- X509V3 Token V1.1

- X509V3 Token V1.0
- X509PKCS7 Token V1.1
- X509PKCS7 Token V1.0
- X509PkiPathV1 Token V1.1
- X509PkiPathV1 Token V1.0
- X509V1 Token V1.1
- Custom Token

The Secure Conversation Token v200502 token type for the WS-Security policy represents the requirement for a Security Context Token as defined in the February 2005 level of the WS-SecureConversation specification.

Enforce token version:

When LTPA Token v2.0 is selected as the token type, both LTPA version 1 and LTPA version 2 tokens can be consumed. Select this checkbox to restrict token consumption to the LTPA Token v2.0 token type.

Local name:

Specifies the local name of the custom token generator or consumer. The **Local name** field is populated based on the token type displayed. Use this field to edit custom token types only.

If the custom token type is used to generate a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile V1.1, use one of the values listed below for the local name. The value you choose depends on the specification level of the Kerberos token generated by the Key Distribution Center (KDC). The following table lists the values and the specification level associated with each value. For purposes of interoperability, the Basic Security Profile V1.1 standard requires the use of the local name http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ.

Local Name Value for Kerberos Token	Associated Specification Level
http://docs.oasis-open.org/wss/oasiswss-kerberos-token-profile-1.1#Kerb erosv5_AP_REQ	Kerberos v5 AP-REQ as defined in the Kerberos specification. Use this value when the Kerberos ticket is an AP Request.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ	GSS-API Kerberos V5 mechanism token containing a KRB_AP_REQ message as defined in RFC-1964 [1964], Sec. 1.1 and its successor RFC-4121, Sec. 4.1. Use this value when the Kerberos ticket is an AP Request (ST + Authenticator).
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510	Kerberos v5 AP-REQ as defined in RFC1510. Use this value when the Kerberos ticket is an AP Request per RFC1510.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510	GSS-API Kerberos V5 mechanism token containing a KRB_AP_REQ message as defined in RFC-1964, Sec. 1.1 and its successor RFC-4121, Sec. 4.1. Use this value when the Kerberos ticket is an AP Request (ST + Authenticator) per RFC1510.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120	Kerberos v5 AP-REQ as defined in RFC4120. Use this value when the Kerberos ticket is an AP Request per RFC4120.

Local Name Value for Kerberos Token	Associated Specification Level
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120	GSS-API Kerberos V5 mechanism token containing an KRB_AP_REQ message as defined in RFC-1964, Sec. 1.1 and its successor, RFC-4121, Sec. 4.1. Use this value when the Kerberos ticket is an AP Request (ST + Authenticator) per RFC4120.

URI:

Specifies the uniform resource identifier (URI) of the custom token generator or consumer. The **URI** field is populated based on the token type displayed. Use this field to edit custom token types only.

Leave this field blank if the custom token type is used to generate a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile V1.1.

JAAS login:

Specifies the Java Authentication and Authorization Service (JAAS) application login information. Click **New** to add a new JAAS application login or JAAS system login entry.

If the server is in a security domain that includes specific system or application logins, these logins are listed in the JAAS login menu, in addition to the global logins.

New Application Login:

Click to go to the effective JAAS login collection for the current security domain.

Custom properties – Name:

Specifies the name of the custom property. Custom properties are not initially displayed in this column until they are added.

Select one of the following actions for custom properties:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Edit	Specifies that you can edit the selected custom property. Select this action to provide input fields and create the listing of cell values for editing. The **Edit** button is not available until at least one custom property has been added.
Delete	Removes the selected property.

If the custom token type is used to generate a Kerberos token, specify the following custom properties:

Custom property name	Value
com.ibm.wsspi.wssecurity.krbtoken.targetServiceName	Specifies the name of the target service.
com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost	Specifies the host name that is associated with the target service.
com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm	Specifies the name of the realm that is associated with the target service.

For the token generator, the combination of the target service name and target hostname forms a Service Principal Name (SPN), which represents the target Kerberos service principal name. The Kerberos client requests the initial Kerberos AP_REQ token for the SPN.

Custom properties – Value:

Specifies the value of the custom property. Use the **Value** field to enter, edit, or delete the value for a custom property.

Callback handler:

After all other configurations on the protection token page are applied or saved, this section is displayed and links to the configuration settings for the callback handler. Click this link to specify callback handler settings that determine how security tokens are acquired from message headers.

Tolerate secure conversation token V200502:

The secure conversation token V200502 token type for the WS-Security policy represents the requirement for a secure conversation token as defined the in the February 2005 level of the WS-SecureConversation specification. This option specifies whether the provider handles both secure conversation token V1.3 and secure conversation token V200502. By default, the provider handles both versions. You can change this behavior by clicking to remove the check box selection so that the provider handles only the V1.3 token.

Note: This checkbox is displayed only in the service provider token consumer panel.

Data type	Check box
Range	Selected or cleared
Default value	Selected

Authentication generator or consumer token settings:

Authentication tokens are used to prove or assert an identity. Use the administrative console to add authentication token settings for message parts when you are editing a general binding.

To configure authentication tokens, complete the following steps:

1. To view and select the general bindings that are set as the global security default policy set bindings, click **Services > Policy sets > Default policy set bindings**. The specified bindings are used unless overridden at the attachment point, at the server, or at a security domain.
2. To access and configure the general bindings and to add authentication token settings for message parts, click **Services > Policy sets > General provider policy set bindings**.
3. Click the **WS-Security** policy in the Policies table.
4. Click the **Authentication and protection** link in the Main message security policy bindings section.
5. Click **New token** to create a new token generator or consumer, or click an existing consumer or generator token link from the Authentication Tokens table.

To view and configure application-specific bindings for tokens and message parts that are required by a policy set, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned an application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.

7. Click a consumer or generator token link from the Protection Tokens table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the name of the token being configured. When using application specific bindings, this field is not displayed.

Token type:

Specifies the type of token being configured.

When you are using application specific bindings, the token type is obtained from the policy file and it is read-only. When you are using general bindings, select a token type from the list. The following token types are available:

- X509V3 Token V1.1
- X509V3 Token V1.0
- Username Token V1.1
- Username Token V1.0
- X509PKCS7 Token V1.1
- X509PKCS7 Token V1.0
- X509PkiPathV1 Token V1.1
- X509PkiPathV1 Token V1.0
- LTPA Propagation Token
- X509V1 Token V1.1
- LTPA Token
- LTPA Token V2.0
- Custom Token

Note: The LTPA Token V2.0 token type is available only for bindings using the new namespace in IBM WebSphere Application Server, Version 7.0 or later. When you select LTPA Token V2.0 as the token type for the token consumer, both LTPA tokens and LTPA V2.0 tokens can be consumed. To restrict the token consumer to LTPA V2.0 tokens only, select the **Enforce token version** check box.

If you select LTPA Token as the token type for the token generator, single sign-on interoperability mode must be enabled. This is a setting in global security from Web and SIP security. If the interoperability flag is not set to enabled (true), an error occurs when the application that is attached to these bindings is started. If you want to use the LTPA token without checking the state of the interoperability flag, you can set the custom property, com.ibm.wsspi.wssecurity.tokenGenerator.ltpav1.pre.v7, on the token generator.

Local name:

Specifies the local name for the authentication token generator or consumer. The **Local name** field is populated based on the token type displayed. Use this field to edit custom token types only.

URI:

Specifies the uniform resource identifier (URI) of the authentication token generator or consumer. The **URI** field is populated based on the token type displayed. Use this field to edit custom token types only.

Leave this field blank if the custom token type is used to generate a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile v1.1.

Security token reference:

Specifies the security token reference. The security token reference field is displayed only for authentication tokens in application-specific bindings. This field is not available for default bindings.

JAAS login:

Specifies a list of application and system Java Authentication and Authorization Service (JAAS) logins that are effective for the domain to which the binding is scoped.

If an application is scoped to the global security or if it is scoped to a domain that does not customize its own JAAS logins, then the list of global logins are displayed in the menu list. Click **New Application Login** to access the global JAAS application login collection. The JAAS login menu list and **New Application Login** button behavior depend on whether the binding is being created in association with an attachment. Use caution when changing security domains, since a previously-referenced security configuration, such as JAAS logins, might not be accessible in a different security domain.

Custom properties – Name:

Specifies the name used for the custom property.

Custom properties are not initially displayed in this column. Click one of the following buttons to enable the actions described:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Edit	Enables the selected custom property to be edited. Clicking this button provides input fields and creates the listing of cell values to be edited. The Edit button is not available until at least one custom property has been added.
Delete	Removes the selected custom property.

Custom properties – Value:

Specifies the value of the custom property to be used. Use the **Value** field to enter, edit, or delete the value for a custom property.

If the custom token type is used to generate a Kerberos token, specify a custom property with the com.ibm.wsspi.wssecurity.krbtoken.targetServiceName target service name, the com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost host name that is associated with the target service, and the com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm Kerberos realm that is associated with the target service. The com.ibm.wsspi.wssecurity.krbtoken.targetServiceName and com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost custom properties are required. The com.ibm.wsspi.wssecurity.krbtoken.targetServiceRealm custom property is optional. For the token generator, the combination of the target service name and target hostname forms a Service Principal Name (SPN) which represents the target Kerberos service principal name. The Kerberos client requests the initial Kerberos AP_REQ token for the SPN.

Callback handler:

Links to the Callback handler page where you can configure callback handlers. Callback handler settings determine how security tokens are acquired from messages headers.

If you are working with a Username token or LTPA token that is using default bindings, the user names and passwords might have been provided as examples. You need to update the values for these token types.

Callback handler settings:

Use this page to configure callback handler settings, which determine how security tokens are acquired from messages headers.

You can configure callback handler settings when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Authentication and protection** link in the Main message security policy bindings section.
4. Click the ***name_of_token*** link in the Protection tokens section or the Authentication tokens section.
5. Click the **Callback handler** link.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned an application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.
7. Click the ***name_of_token*** link in the Protection tokens section or the Authentication tokens section.
8. Click the **Callback handler** link.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

The Callback Handler displays fields differently for different tokens being configured. Depending on whether you are configuring generator or consumer tokens for protection or you are configuring inbound or outbound tokens for authentication, the sections and fields on this panel display some or all of the fields explained in this topic, as noted in the description of each field.

Class name:

The fields in the Class name section are available for all types of token configuration.

Select the class name to use for the callback handler. Select the **Use built-in default** option for normal operation. Use the **Use custom** option only if you are using a custom token type.

For the Kerberos custom token type, use the class name, com.ibm.websphere.wssecurity.callbackhandler.KRBTokenGenerateCallbackHandler, for token generator configuration. Use com.ibm.websphere.wssecurity.callbackhandler.KRBTokenConsumeCallbackHandler for token consumer configuration.

Use built-in default:

Specifies that the default value is used for the class name. Use the default value (shown in the field) for the class name when you select this radio button. This name is based on the token type and whether the callback handler is for a token generator or a token consumer. This option is mutually exclusive to the **Use custom** option.

Use custom:

Specifies that a custom value is used for the class name. Select this radio button and enter the name in the field to use a custom class name.

No default value is available for this entry field. Use the information in the following table to determine this value:

Token Type	Consumer or Generator	Callback Handler Class Name
UsernameToken	consumer	com.ibm.websphere.wssecurity.callbackhandler.UNTConsumeCallbackHandler
UsernameToken	generator	com.ibm.websphere.wssecurity.callbackhandler.UNTGenerateCallbackHandler
X509Token	consumer	com.ibm.websphere.wssecurity.callbackhandler.X509ConsumeCallbackHandler
X509Token	generator	com.ibm.websphere.wssecurity.callbackhandler.X509GenerateCallbackHandler
LTPAToken/LTPAPropagationToken	consumer	com.ibm.websphere.wssecurity.callbackhandler.LTPAConsumeCallbackHandler
LTPAToken/LTPAPropagationToken	generator	com.ibm.websphere.wssecurity.callbackhandler.LTPAGenerateCallbackHandler
SecureConversationToken	consumer	com.ibm.ws.wssecurity.impl.auth.callback.SCTConsumeCallbackHandler
SecureConversationToken	generator	com.ibm.ws.wssecurity.impl.auth.callback.WSTrustCallbackHandler

This button is mutually exclusive to the **Use built-in default** option.

Certificates:

The fields in the Certificates section are available if you are configuring a protection token. For a consumer token, you can use the Trust any certificate or the Certificate store options to configure the certificate. For a generator token, you can click a certificate from the listing or click the **New** button to add one.

Certificates - Trust any certificate:

If the protection token has a certificate configured, this option specifies that the system will trust any certificate, and not define the certificate store. Select this option to trust each certificate. This option is mutually exclusive to the **Certificate store** option and is only applicable to the token consumer.

Certificates - Certificate store:

Specifies, if the protection token has a certificate configured, the certificate store to be trusted. Select this option to trust each certificate store specified in the entry field. This option is mutually exclusive to the **Trust any certificate** option. When you select this option, the **New** button is enabled so that you can configure a new certificate store. You can also add a second certificate store to the **Trusted anchor store** entry field when you click **Certificate store**. The **Trusted anchor store** field is only applicable to the token consumer.

Basic authentication:

The fields in the Basic authentication section are available if you are configuring an authentication token that is not an LTPA propagation token.

For the Kerberos custom token type, you must complete the Basic Authentication section for the Kerberos login.

User name:

Specifies the user name that you want to authenticate.

Password:

Specifies the password to be authenticated. Enter a password to authenticate in this entry field.

Confirm password:

Specifies the password that you want to confirm.

Keystore:

The fields in the Keystore section are available if you are configuring a protection token.

In the Keystore name list, you can click **Custom** to define a custom keystore, click one of the externally defined keystore names, or click **None** if no keystore is required.

Keystore - Name:

Specifies the name of the centrally managed keystore file that you want to use.

Click the name of a centrally managed keystore name from this menu or enter one of the following values:

NodeDefaultKeyStore
NodeDefaultTrustStore
NodeLTPAKeys
None Specifies to not use a centrally managed keystore file.

Custom
Specifies to use the centrally managed keystore file. Click the **Custom keystore configuration** link to configure custom keystore and key settings.

Keystore - Custom keystore configuration:

Specifies a link to create a custom keystore. Click this link to open a panel where you can configure a custom keystore.

Key:

The fields in the Key section are available if you are configuring a protection token.

Name:

Specifies the name of the key to use. Enter the name of the key to be used in this required field.

Alias:

Specifies the alias name of the key that you want to use. Enter the alias of the name of the key to use in this required field.

Password:

Specifies the password for the key that you want to use.

You cannot set a password for public keys for asymmetric encryption generator or asymmetric signature consumer.

Confirm password:

Specifies the confirmation of the password for the key that you want to use. Enter the password that you entered in the Password field to confirm.

Do not provide a key confirm password for public keys for asymmetric outbound encryption or inbound signature.

Custom properties:

The fields in the Custom properties section are available for all types of token configuration.

You can add custom properties needed by the callback handler here using name-value pairs.

To implement signer certificate encryption when using the JAX-WS programming model, add the custom property **com.ibm.wsspi.wssecurity.token.cert.useRequestorCert** with the value true on the callback handler of the encryption token consumer. This implementation uses the public key from the certificate of the signer. This custom property is used by the response generator.

For a Kerberos custom token based on OASIS Web Services Security Specification for Kerberos Token Profile V1.1, specify the following property for token generation: **com.ibm.wsspi.wssecurity.krbtoken.clientRealm**. This allows the Kerberos client realm to initiate the Kerberos login. If not specified, the default Kerberos realm name is used.

Name:

Specifies the name of the custom property to use.

Custom properties are not initially displayed in this column. Click one of the following actions for custom properties:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Delete	Removes the selected custom property.

Value:

Specifies the value of the custom property to use. With the **Value** entry field, you can enter or delete the value for a custom property.

Custom keystore settings:

Use this page to configure custom keystore files. Custom keystore files are alternatives to the key management support built into the WebSphere Application Server. The callback handler uses the custom version of the keystore configuration that includes keys.

You can configure custom keystore files for message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.

3. Click the **Authentication and protection** link in the Main message security policy bindings section.
4. Click a ***protection_token*** link in the Protection tokens table.
5. Click the **Callback handler** link in the Additional bindings section.
6. Select **Custom** from the list in the Keystore section.
7. Click the **Custom keystore configuration** link.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Authentication and protection** link in the Main message security policy bindings section.
7. Click a ***protection_token*** link in the Protection tokens table.
8. Click the **Callback handler** link in the Additional bindings section.
9. Select **Custom** from the list in the Keystore section.
10. Click the **Custom keystore configuration** link.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Keystore:

Use this section to specify information about the custom keystores.

Full path:

Specifies the full path to where the keystore file is located. Enter the path to the keystore file in this required field. You can use system variables for portions of the path. For example you might enter ${USER_INSTALL_ROOT}/etc/ws-security/myKeyStore.jks. This field is required for the custom keystore configuration.

Type:

Specifies the type of the keystore file to use.

Password:

Specifies the password to use.

Confirm password:

Specifies the password to be use and confirms the one entered in the **Password** field.

Key:

Use this section to specify information about the key.

Name:

Specifies the name of the key to use. Enter the name of the key to be used in this required field.

Alias:

Specifies the alias name of the key that you want to use. Enter the alias of the name of the key to use in this required field.

Password:

Specifies the password for the key that you want to use.

You cannot set a password for public keys for asymmetric signature inbound and encryption outbound. The Password and Confirm Password fields display only for the following:

client	asymmetric signature outbound	AsymmetricBindingInitiatorSignatureToken0
client	asymmetric encryption inbound	AsymmetricBindingInitiatorEncryptionToken0
server	asymmetric signature outbound	AsymmetricBindingRecipientSignatureToken0
server	asymmetric encryption inbound	AsymmetricBindingRecipientEncryptionToken0

Confirm password:

Specifies the confirmation of the password for the key that you want to use. Enter the password that you entered in the Password field to confirm.

Similar to the Password field, you cannot confirm the password for public keys for asymmetric signature inbound and encryption outbound.

Caller settings

Use this page to configure the caller settings. The caller specifies the token or message part that is used for authentication.

You can configure the caller settings for message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a general provider binding, complete the following actions:

1. Click **Services > Policy sets > General provider policy sets bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Authentication and protection** link in the Main message security policy bindings section.
4. Click the **Caller** link in the Main message security policy bindings section.
5. Click **New**.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link in the Web Services Properties section. The caller settings are available only for the service provider policy sets and bindings. The caller settings are not available for service client policy sets and bindings.

4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Caller** link in the Main message security policy bindings section.
7. Click **New**.

   Note: When you create a new caller it will automatically be assigned the next available order. You can change the order of preference, as described in the Order section below.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Name:

Specifies the name of the caller to use for authentication. Enter a caller name in this required field. This arbitrary name identifies this caller setting.

Order:

Specifies the order of preference for the callers. The order determines which caller will be utilized when multiple authentication tokens are received.

You can change the order of preference by moving a caller up or down in the list. Click the checkbox next to a caller name to select the caller, then click the **Move up** button to move the caller higher in the list, or click the **Move down** button to move the caller to a lower position in the preference order.

Button	Resulting Action
Move up	Moves the order of the selected caller up in the caller list.
Move down	Moves the order of the selected caller down in the caller list.

Note: The order column displays only for bindings using the new namespace. If a binding with multiple callers was migrated to the new namespace, then the callers do not have an order. In that case, an error message is displayed. When this occurs, select a caller in the table and then click either **Move up** or **Move down** to assign an order to each caller. Callers must have orders assigned before you save the bindings or use the bindings with an application.

Caller identity local part:

Specifies the local name of the caller to use for authentication. Enter a caller identity local name in this required field.

When specifying an LTPA caller, use LTPA as the local name for a caller that uses an older binding, prior to IBM WebSphere Application Server, Version 7.0. Newer bindings for IBM WebSphere Application Server, Version 7.0 and later should use LTPAv2 as the local name. Specifying LTPAv2 allows both LTPA and LTPAv2 tokens to be consumed, unless the Enforce token version option is selected on the token consumer.

Caller identity URI:

Specifies the uniform resource identifier (URI) of the caller to use for authentication. Enter a caller URI in this field.

When specifying an LTPA caller, use http://www.ibm.com/websphere/appserver/tokentype/5.0.2 as the URI for a caller that uses an older binding, prior to IBM WebSphere Application Server, Version 7.0. Newer

bindings for IBM WebSphere Application Server, Version 7.0 and later should use the http://www.ibm.com/websphere/appserver/tokentype URI.

Use identity assertion:

Specifies whether identity assertion is used when authenticating. Select this check box if you want to use identity assertion. Select this box to enable the **Trusted identity local name** and **Trusted identity URI** fields.

Trusted identity local name:

Specifies the trusted identity local name when the identity assertion is used. Enter a trusted identity local name in this entry field when the identity assertion is used.

Trusted identity URI:

Specifies the trusted identity uniform resource identifier (URI). Enter a URI in this entry field when the identity assertion is used.

Signing part reference:

When the trusted identity is based on a signing token, select the signing part reference that represents the message parts signed by that token.

Callback handler:

Specifies the class name of the callback handler. Enter the class name of the callback handler in this field.

JAAS login:

Specifies the Java Authentication and Authorization Service (JAAS) application login. You can enter a JAAS login, select one from the menu, or click **New** to add a new one.

Custom properties – Name:

Specifies the name of the custom property.

Custom properties are not initially displayed in this column. Select one of the following actions for custom properties:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Edit	Specifies that you can edit the custom property value. At least one custom property must exist before this option is displayed.
Delete	Removes the selected custom property.

Custom properties – Value:

Specifies the value of the custom property that you want to use. Use the Value field to add, edit, or delete the value for a custom property.

Caller collection

The caller specifies the token or message part that you want to use for authentication. Use this administrative console page to access, view and configure the caller settings for message parts.

To configure general bindings for tokens and message parts that are required by the policy set, complete the following steps.

1. To access and configure the general bindings, click **Services > Policy sets > General provider policy set bindings**. The caller settings are available only for the service provider policy sets and bindings. The caller settings are not available for service client policy sets and bindings.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Caller** link in the Main message security policy bindings section.

To view and configure application specific bindings for tokens and message parts that are required by a policy set, complete the following steps:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider.
3. Click the **Service provider policy sets and bindings** link in the Web Services Properties section. The caller settings are available only for the service provider policy sets and bindings. The caller settings are not available for service client policy sets and bindings.
4. Select a binding. You must have previously attached a policy set and assigned an application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Caller** link in the Main message security policy bindings section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Name:

Specifies the name of the caller to use for authentication. Select a caller name from this field.

The following actions are available to work with callers.

Button	Resulting Action
New	Opens the Caller settings page, which you use to add a caller.
Delete	Removes the selected caller.

Order:

This number specifies the order of preference for the configured callers. If multiple caller tokens are found in an incoming message, the caller used for authentication will be the one with highest priority, based on decreasing order of preference.

You can change the order of preference using the **Move Up** and **Move Down** buttons.

Button	Resulting Action
Move Up	Moves the selected caller up in the order of preference, switching positions with the immediately preceding caller. The selected caller is now preferred over the caller that you demoted in the list.
Move Down	Moves the selected caller lower in the order of preference, switching positions with the caller following it. The demoted caller is now lower in preference than the caller that was previously below it.

Caller Identity Local Part:

Specifies the local identity part of the caller to use for authentication.

Caller Identity URI:

Specifies the uniform resource identifier (URI) of the caller to use for authentication.

Message expiration settings

Use this page to define settings for message expiration, if and when messages expire. When you specify message expiration, the message expires after the specified interval of time passes.

You can define message expiration settings for tokens and message parts when you are editing a default cell or server binding. You can also configure application specific bindings for tokens and message parts that are required by the policy set.

To view this administrative console page when you are editing a default cell binding, complete the following actions:

1. Click **Services > Policy sets > Default policy set bindings**.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Message expiration** link in the Main message security policy bindings section.

To view this administrative console page when you are configuring application specific bindings for tokens and message parts that are required by the policy set, complete the following actions:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. You must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Message expiration** link in the Main message security policy bindings section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Message expiration:

Specifies whether message expiration is enabled. To enable message expiration, select this check box. Leave it unchecked to disable message expiration.

Message timeout interval:

Specifies the time, in minutes, for the message to time out if message expiration is enabled. This field is enabled only when you select the **Enable message expiration** check box.

Actor roles settings

Use this page to define settings for SOAP actor roles. The SOAP actor, also known as the SOAP role, defines the intermediary or ultimate recipient of a message.

To view this administrative console page use one of the following options:

1. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select an application that contains Web services. The application must contain a service provider or a service client.

3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** in the Web Services Properties section.
4. Select a binding. The binding must have previously attached a policy set and assigned a application specific binding.
5. Click the **WS-Security** policy in the Policies table.
6. Click the **Actor roles** link in the Main message security policy bindings section.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Inbound actor role URI:

Specifies the name of the uniform resource identifier (URI) for the inbound actor role.

Outbound actor role URI:

Specifies the name of the uniform resource identifier (URI) for the outbound actor role.

Web Services Addressing policy set binding

Use this page to modify the endpoint reference binding for Web Services Addressing (WS-Addressing). The product enforces this binding on JAX-WS Web service applications that use WS-Addressing. This panel applies only to the Network Deployment version of the product.

To view this administrative console page, click **Services** → **Policy Sets** → **Default policy set bindings** → **WS-Addressing**.

Prevent workload management of referenced endpoints in clusters:

By default, when an application uses the WS-Addressing API createEndpointReference(service, port) method to create an endpoint reference in a cluster environment, that endpoint reference is workload managed. Select this option to prevent the workload management of such endpoint references.

If you select this option, messages that are targeted at such an endpoint reference are always sent to the node on which the endpoint reference was created. This behavior is useful when the endpoint contains in-memory state that is held on a single node, for example, in Web Services Resource Framework (WS-RF) applications.

Note: Although the endpoint reference always represents the same endpoint, the state that is held at that endpoint is not reliable. If the node that is hosting the endpoint fails, the state could be lost. Note also that if the node fails, the endpoint reference is no longer valid.

Data type Check box

Default Cleared

Range

Cleared
Endpoint references that are created by the application in a cluster environment are workload managed.

Selected
Endpoint references that are created by the application in a cluster environment are not workload managed.

Attaching a policy set to a service artifact

Attach a policy set to a service artifact, such as an application, service, or endpoint to define the quality of services that are supported. Policy sets can define the policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport.

Before you begin

Before you can start this task, you must deploy an application containing Web services. Also, if none of the default policy sets contain the necessary policy definitions, then you must create a custom policy set with the necessary definitions.

About this task

Develop a Web service that contains each of the necessary artifacts, and deploy your Web services application into your application server instance. Now you can attach policy sets to your service artifacts, such as an application, service, or endpoint. To attach a policy set to a service artifact perform the following steps:

1. Open the administrative console.
2. To attach a policy set to a service provider, click **Applications > Enterprise applications** > ***application_name*** > **Service provider policy sets and bindings**.

   To attach a policy set to a service client, click **Applications > Enterprise applications** > ***application_name*** > **Service client policy sets and bindings**.
3. Select the check box for the service artifact.
4. Select **Attach** to display a list of available policy sets to attach. Select a policy set from the list.
5. Click **Save**, to save your changes to the master configuration.
6. [Optional] To see what attachments are defined for a given policy set, select **Services** > **Policy sets** > **Application policy sets** > ***policy_set_name*** > **Attached applications**.

Results

When you finish these steps, a policy set is attached to the service artifact.

Example

If you have the application, `app1` and you want to attach the policy set, `WSSecurity default`, then perform the following steps:

1. Locate the `app1` application in the **Applications** > **Enterprise applications** collection.
2. Click the **`app1`** application.
3. Click the **Service provider policy sets and bindings** link or the **Service client policy sets and bindings** link.
4. Select the check box for the service artifact where the policy set is to be attached.
5. Click **Attach**. Select **`WSSecurity default`** policy set.
6. Click **Save**, to save your changes to the master configuration.

What to do next

You can create custom bindings for policy set attachments. Read about creating custom bindings for policy set attachments.

You can configure the service client or service provider to share their policies. Read about using WS-Policy to exchange policies in a standard format.

Related tasks

"Creating application specific bindings for policy set attachment" on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

"Reassigning bindings to policy sets" on page 856
After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

"Modifying default bindings at the server level for policy sets" on page 855
You can define default bindings for HTTP transport, JMS transport, Secure Sockets Layer (SSL) transport, WS-Addressing, WS-ReliableMessaging, and WS-Security policies.

"Using WS-Policy to exchange policies in a standard format" on page 714
WS-Policy is an interoperability standard that is used to describe and communicate the policies of a Web service so that service providers can export policy requirements in a standard format. Clients can combine the service provider requirements with their own capabilities to establish the policies required for a specific interaction. WebSphere Application Server conforms to the WS-Policy specification, so that policy information can be exchanged and received in accordance with the WS-Policy standard.

Managing policies in a policy set using the administrative console

When working with policy sets in the administrative console, you can customize the included policies to ensure message security. You can enable, disable, customize, add, or delete policies from a policy set. With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

Before you begin

You can customize the policies for custom policy sets. The provided default policy sets cannot be edited. Make sure that you have created a copy of the default policy set or created a completely new policy set to specify the policies for this policy set.

About this task

Customize policies associated with a policy set using the administrative console.

1. Click one of the following:
   - **Services > Policy sets > Application policy sets** >*policy_set_name*
   - **Services > Policy sets > System policy sets** >*policy_set_name*

   You can also click **Services > Policy sets > Application policy sets** > **New** or **Services > Policy sets > System policy sets** > **New**. Follow this path when you want to create a new policy set and the associated policy or policies.
2. Add a policy to a custom policy set. To do this, see "Adding policies to policy sets using the administrative console" on page 903.
3. Enable a policy for a custom policy set. To do this, see "Enabling policies for policy sets using the administrative console" on page 950.
4. Optional: You can also disable a policy from a custom policy set. To do this, see "Disabling policies from policy sets using the administrative console" on page 951.
5. Optional: You can also delete a policy from a custom policy set. To do this, see "Deleting policies from policy sets using the administrative console" on page 904.

Results

After you have customized your policies, the associated policy set can protect messages according to the policy or policies defined.

What to do next

If the policy set you have modified is an attached policy set, restart all affected applications to pick up the changes you made. If the policy set is unattached, then no further action is required.

Adding policies to policy sets using the administrative console

You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

Before you begin

Before adding a policy to a policy set, check that it is not listed as disabled in the **Policies** table. A disabled policy can be listed for the policy set and only need to be enabled to be included in the policy set. If this is the case, the policy does not need to be added and it is not available to be added.

About this task

All policies are initially set to their default values. These values can be edited and the attributes changed using the administrative console.

1. Click **Services > Policy sets > Application policy sets** > ***policy_set_name*** or **Services > Policy sets > System policy sets > *policy_set_name***. This displays a listing of available policies in the **Policies** table for the policy set selected. If you created a new policy set and did not copy and rename an existing policy set, this table might contain no policies and you must add them.
2. Add a policy. To add a policy not listed in the **Policies** table of the Policy set settings page, click the **Add** drop down button. This button displays a list of available policies that are not already listed in the **Policies** table and therefore not included in the policy set.
3. Click the policy to be added. The selected policy is added to the **Policies** table. Each policy you add has default settings that can be modified.

Results

After you have added a policy, it is then included in the policy set.

Example

If you were working on policy set ABC_ps (that you created and it therefore has no policies associated with it) you would click the ABC_ps name in the **Policy Sets** collection and then click the **Add** button in the **Policies** table on the **Policy set** settings window. The drop down list displays all available policies for you to choose from. You might decide you want to include the WS-Addressing and WS-Security policies for this policy set. You would select each of those from the list. Then, click the policy name in the table to edit the properties of these policies.

What to do next

If the modified policy set is an attached policy set, restart all affected applications to apply the changes. If the policy set is unattached, then no further action is required.

Related tasks

“Deleting policies from policy sets using the administrative console”
You can use the administrative console to delete policies from policy sets. Deleting the policy removes the policy that further defined the rules governing the policy set.

“Enabling policies for policy sets using the administrative console” on page 950
Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

“Disabling policies from policy sets using the administrative console” on page 951
You can have policies listed in a policy set that are in the enabled state so that they are currently included in the policy set. You can disable a policy from being included in a policy set without deleting it from the policy set. You might want to do this if you want the policy included in the policy set in the future. You can use the administrative console to change this setting.

“Web services policies” on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Deleting policies from policy sets using the administrative console

You can use the administrative console to delete policies from policy sets. Deleting the policy removes the policy that further defined the rules governing the policy set.

Before you begin

Before deleting a policy from a policy set, be sure that you no longer want the policy to be available to the policy set. Otherwise, consider disabling it so that it is still available to the policy set but not currently associated with the policy set. You can disable the policy and leave it available for the policy set by clicking the **Select** button by the policy to be disabled and then clicking the **Disable** button. That policy (and any configuration changes you have made to it) is still available to the policy set but is not currently included in the policy set.

About this task

To delete a policy from a policy set, use the administrative console.

1. Click **Services > Policy sets > Application policy sets** > ***policy_set_name*** or **Services > Policy sets > System policy sets > *policy_set_name***. The Policy set settings page displays a listing of available policies in the **Policies** table of the Policy set settings page for the policy set selected. If you created a new policy set and did not copy and rename an existing policy set, this table might contain no policies to delete.
2. Delete the policy from the policy set.
   a. Click the **Select** box beside the policy that you want to delete in the **Policies** table of the Policy set settings page.

b. Click the **Delete** button.

3. Verify that you want to delete the selected policy from the policy set.

Results

You have deleted the policy from the policy set.

Example

If you have a policy set containing a policy that you no longer want included in that policy set, you have two options: disable it or delete it. In some cases, you might want to disable the policy if you have configured attributes for it that you might want to enable at a later time. You can disable the policy and leave it available for the policy set by clicking the **Select** button by the policy to be disabled and then clicking the **Disable** button. But for this example, you have decided that the policy WS-Addressing is no longer valid for your custom policy set ABC_ps and you and you want to delete it. To delete it, click the **Select** button beside the WS-Addressing policy and then click **Delete**. Confirm that you want to delete the policy.

What to do next

You can now add other policies to the policy set or enable or disable existing policies.

Related tasks

"Adding policies to policy sets using the administrative console" on page 903
You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

"Enabling policies for policy sets using the administrative console" on page 950
Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

"Disabling policies from policy sets using the administrative console" on page 951
You can have policies listed in a policy set that are in the enabled state so that they are currently included in the policy set. You can disable a policy from being included in a policy set without deleting it from the policy set. You might want to do this if you want the policy included in the policy set in the future. You can use the administrative console to change this setting.

"Web services policies" on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

"Removing policy set attachments using the wsadmin tool" on page 543
You can use the Jython or Jacl scripting language to remove and transfer policy sets from application artifacts. You can also remove resources that apply to a policy set attachment without deleting the policy set attachment.

Related reference

"Application policy sets collection" on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

"Application policy set settings" on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Modifying policies using the administrative console

With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

Before you begin

The provided default policy sets cannot be edited. You can create a copy of the default policy set or create a completely new policy set in order to specify the policies for it.

About this task

Polices are associated with policy sets, so you can change an instance of a policy for a particular policy set.

1. Select the policy set containing the policy you want to modify. To customize the policies associated with a policy set, from the administrative console, click:
   - **Services > Policy sets > Application policy sets** >*policy_set_name*. Click a policy name in the Policies table.
   - Or **Services > Policy sets > System policy sets** >*policy_set_name*. Click a policy name in the Policies table.
2. Modify the settings. Depending on which Policy you choose, different settings can be modified.
3. [Optional] If the policy to be modified is not already in the Policies table, click **Add** and select a policy from the list to modify.
4. Save the changes you have made. Once you change the settings on a policy, you need to save the changes to return to the policy set.

Results

The policy set configuration is saved with the selected modifications.

Example

You have created a copy of the WSReliableMessaging persistent policy set and named it `WSRM_p1`. You want to change the settings on the WSReliableMessaging policy that is included, by default, in the copy of this policy set. So you click your `WSRM_p1` policy set from the Application policy sets window and then click the WSReliableMessaging policy from the Policies table. You can then alter the following settings:

Standard
The default setting is WSReliableMessaging 1.1.

Deliver messages in the order that they were sent
The default setting is false. Valid values are true or false.

Quality of service
The default is Unmanaged non-persistent

Enable 'MakeConnection' for synchronous two-way message exchange
This setting is selected by default.

Save your changes and return to the Application policy sets window for the WSRM_p1 policy set.

What to do next

You can use the policy set as you have configured or you can change the bindings, attach or detach the policy set from applications.

Note: Because this policy set specifies managed persistent quality of service, you need to define bindings to the service integration bus and messaging engine that you want to use to manage the WS-ReliableMessaging state. For more information, see "Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console" on page 657 or using the wsadmin tool.

Configuring the WS-ReliableMessaging policy:

When working with policy sets in the administrative console, you can customize some policies.

Before you begin

This task assumes that you are working with a policy set to which the WS-ReliableMessaging policy has been added.

Do not edit the policies associated with the provided default policy sets. If you need to modify the reliable messaging policy settings, use a copy of a default policy set or create a new policy set.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

About this task

To configure the WS-ReliableMessaging policy for a given policy set, use the administrative console to complete the following steps:

1. In the navigation pane, click **Services** → **Policy sets** → **Application policy sets** → ***policy_set_name*** → **WS-ReliableMessaging**. The "WS-ReliableMessaging settings" on page 908 form is displayed.
2. Modify one or more of the following properties:

   Standard

   Select the WS-ReliableMessaging specification to use for reliable transmission of your messages. WS-ReliableMessaging Version 1.1 is the default value. Details of the supported WS-ReliableMessaging specifications are available at the following Web addresses:

   - The WS-ReliableMessaging specification Version 1.0, February 2005.
   - The OASIS WS-ReliableMessaging specification Version 1.1, February 2007.

   Note: If you plan to invoke a .NET-based Web service, you must select WS-ReliableMessaging Version 1.0.

   Deliver messages in the order that they were sent

   Select this option if the sender of a request has to receive a response before it sends the next request, or if you want to enable transaction support for inbound (provider) message exchanges as described in "Providing transactional recoverable messaging through WS-ReliableMessaging" on page 653, or if you want to marginally increase reliability as described in "A message is not recovered after a server becomes unavailable, even with the managed persistent quality of service" on page 668.

   Note: When you enable this option, WS-ReliableMessaging ensures that the messages are made available to the requester application in the order that they were sent. That is, if WS-ReliableMessaging cannot make a given message available, it will not make any subsequent messages available. However, the requester application must also poll for the messages in the order in which it wants to receive them. For example:

   a. WS-ReliableMessaging makes message 1 available, then message 2, then message 3.
   b. The requester application uses asynchronous polling to deliberately poll for message 2, then message 3, then message 1. All three messages are available, so this polling out of order is successful.

   Even though WS-ReliableMessaging is delivering the messages in the order that they were sent, the requester application is choosing to receive them out of order.

Quality of service

Click a radio button to choose the required quality of service. The three choices are:

Unmanaged non-persistent

You can configure Web service applications to use WS-ReliableMessaging with a default in-memory store. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages.

Managed non-persistent

This in-memory quality of service option uses a messaging engine to manage the sequence state, and messages are written to disk if memory is low. This quality of service allows for the re-sending of messages that are lost in the network, and can also recover from server failure. However, state is discarded after a messaging engine restart so in this case you will lose messages.

Managed persistent

This quality of service for asynchronous Web service invocations is recoverable. This option also uses a messaging engine and message store to manage the sequence state. Messages are persisted at the Web service requester server and at the Web service provider server, and are recoverable if the server becomes unavailable. Messages that have not been successfully transmitted when a server becomes unavailable can continue to be transmitted after the server restarts.

The default is unmanaged non-persistent.

Note: All three qualities of service are supported when applications are deployed to the application server. Thin client and client container applications use the first option only.

3. Click **OK**.
4. Save your changes to the master configuration.

Results

After you have customized the reliable messaging policy, the associated policy set uses this policy to help ensure reliable delivery of messages.

WS-ReliableMessaging settings:

For the WS-ReliableMessaging policy you can configure the version of the WS-ReliableMessaging standard that you want to use, the order in which messages are delivered, and the required quality of service (the reliability level) for message delivery. The product can enforce these policies on inbound messages and applies them to outbound messages.

To view this pane in the console, click the following path: **Services** → **Policy sets** → **Application policy sets** → ***policy_set_name*** → **WS-ReliableMessaging**.

With WebSphere Application Server, you can use WS-ReliableMessaging with Java API for XML-Based Web Services (JAX-WS) 2.1 Web services applications that use a SOAP over HTTP binding. To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent.

Do not edit the policies associated with the provided default policy sets. If you need to modify the reliable messaging policy settings, use a copy of a default policy set or create a new policy set.

At any stage - that is, before or after you have built your reliable Web service application, or configured your policy sets - you can set a property that configures endpoints to only support clients that use reliable messaging. This setting is reflected by WS-Policy if engaged.

Standard: Select the WS-ReliableMessaging specification to use for reliable transmission of your messages. WS-ReliableMessaging Version 1.1 is the default value. Details of the supported WS-ReliableMessaging specifications are available at the following Web addresses:

- The WS-ReliableMessaging specification Version 1.0, February 2005.
- The OASIS WS-ReliableMessaging specification Version 1.1, February 2007.

Note: If you plan to invoke a .NET-based Web service, you must select WS-ReliableMessaging Version 1.0.

Deliver messages in the order that they were sent:

Select this option if the sender of a request has to receive a response before it sends the next request.

If you enable in-order delivery, you must also ensure that the requester application polls for the messages in the order in which it wants to receive them. For more information, see “Configuring the WS-ReliableMessaging policy” on page 907.

Specifying in-order delivery also marginally increases reliability if you are using the managed persistent quality of service. For more information, see “A message is not recovered after a server becomes unavailable, even with the managed persistent quality of service” on page 668.

Quality of Service: Select one of the following qualities of service:

Unmanaged non-persistent - Tolerates network and remote system failures
You can configure Web service applications to use WS-ReliableMessaging with a default in-memory store. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. The default is Unmanaged Non-Persistent.

Managed non-persistent - Tolerates system, network, and remote system failures, but state is discarded after messaging engine restart
This in-memory quality of service option uses a messaging engine to manage the sequence state, and messages are written to disk if memory is low. This quality of service allows for the re-sending of messages that are lost in the network, and can also recover from server failure. However, state is discarded after a messaging engine restart so in this case you will lose messages.

Managed persistent - Tolerates system, network, and remote system failures
This quality of service for asynchronous Web service invocations is recoverable. This option also uses a messaging engine and message store to manage the sequence state. Messages are persisted at the Web service requester server and at the Web service provider server, and are recoverable if the server becomes unavailable. Messages that have not been successfully transmitted when a server becomes unavailable can continue to be transmitted after the server restarts.

Note:

- All three qualities of service are supported when applications are deployed to the application server. Thin client and client container applications use the first option only.
- For the unmanaged non-persistent quality of service, the messages are stored only in memory. For both of the managed qualities of service, the messages are managed by a messaging engine backed by a message store. You specify a binding to a bus and messaging engine on the

"WS-ReliableMessaging policy binding" form. If your chosen quality of service is Unmanaged Non-Persistent, which does not use a binding to a messaging engine, then any binding that you specify is ignored.

WS-ReliableMessaging policy binding:

To configure a Web service application to use WS-ReliableMessaging, you attach a policy set that contains a WS-ReliableMessaging policy type. This policy type offers a range of qualities of service: managed persistent, managed non-persistent, or unmanaged non-persistent. The managed qualities of service, managed persistent and managed non-persistent, are supported by the service integration bus. Use this page to select the bus and messaging engine to use for the reliable messaging protocol state.

To view this pane in the console, click one of the following paths:

- **Services** → **Policy sets** → **Default policy set bindings** → **Version 6.1 default policy set bindings** → **WS-ReliableMessaging** (This is the default binding for client and provider policy set attachments within WebSphere Application Server Version 6.1 applications, and attachments to service applications that are deployed to a Version 6.1 server.)
- **Services** → **Policy sets** → **General provider policy set bindings** → ***provider_policy_set_binding_name*** → **WS-ReliableMessaging** (This binding is used for the specified provider policy set.)
- **Services** → **Policy sets** → **General client policy set bindings** → ***client_policy_set_binding_name*** → **WS-ReliableMessaging** (This binding is used for the specified client policy set.)

Note:

- You only need to specify a binding to a bus and messaging engine if you are using a managed quality of service. If your chosen quality of service is Unmanaged Non-Persistent, any binding that you specify is ignored. The quality of service is defined on the "WS-ReliableMessaging settings" on page 908 form for your chosen policy set. For more information, see "Configuring the WS-ReliableMessaging policy" on page 907.
- When many applications use the same messaging engine, it can impact performance. Factors to consider include the number of applications that are already binding to the messaging engine, the CPU utilization, and the message throughput. To improve performance for a single server configuration, create a new messaging engine to bind to your application.

Bus name:

Specifies a list of available service integration buses in the cell. Use the list to select a bus, or click **Manage buses, bus members, and messaging engines** to add a new bus. The bus that you add is selected for this binding configuration when you return to this panel.

Messaging engine name:

Specifies a list of each bus member for the selected bus. Use the list to select a bus member, or click **Manage buses, bus members, and messaging engines** to add a new bus member. The bus member that you add is selected for this binding configuration when you return to this panel.

Configuring the WS-Addressing policy:

When working with policy sets in the administrative console, you can add and configure policies to enable standard addressing of Web services.

Before you begin

You can specify policies for custom policy sets. The provided default policy sets cannot be edited. You must create a copy of the default policy set or create a completely new policy set in order to specify the policies for it.

About this task

Adding a WS-Addressing policy enables the support for WS-Addressing. This support provides a standard way to address Web services and include addressing information in messages. Adding a WS-Addressing policy is equivalent to configuring the WSDL file for the Web service to specify that WS-Addressing should be used.

To specify or configure the policies associated with a policy set, use the administrative console.

1. In the navigation pane of the administrative console, click **Services Policy sets** → **Application policy sets** > ***policy_set_name*** → **[Policies] WS-Addressing**. The WS-Addressing settings pane is displayed.
2. Select **WS-Addressing is mandatory** to specify that WS-Addressing information must be included in SOAP message headers. For servers, this setting means that the server returns a fault if it receives a message that does not contain a WS-Addressing header. For clients, this setting means that WS-Addressing headers are always added to SOAP messages. If you have enabled WS-Policy, this requirement is communicated between servers and clients that support WS-Policy.
3. In the **Messaging style** box, select the message exchange pattern to use:
   - **Synchronous and asynchronous**. The targeting of response messages is not restricted.
   - **Synchronous only**. Response messages must be targeted at the WS-Addressing anonymous URI.
   - **Asynchronous only**. Response messages must not be targeted at the WS-Addressing anonymous URI.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

After you have included the WS-Addressing policy in a policy set, the associated policy set uses this policy to address Web services.

WS-Addressing policy settings:

Use this panel to define the appropriate WS-Addressing policy assertions for this policy set.

To view this administrative console page, click **Services** → **Policy sets** → **Application policy sets** → ***policy_set_name*** → **[Policy] WS-Addressing**, when the policy set includes the WS-Addressing policy type.

You can configure the WS-Addressing policy type for both client-side and provider-side policy sets. If you enable WS-Policy, this configuration is communicated between servers and clients that support WS-Policy.

WS-Addressing is mandatory:

Specifies whether a WS-Addressing SOAP header is included on messages.

Data type	Check box
Default	Cleared

Range

Cleared
WS-Addressing is not mandatory. Servers will not generate a fault if they receive a message that does not contain a WS-Addressing header. Clients might not include WS-Addressing headers in SOAP messages, for example, if WS-Policy is enabled and the server does not specify that WS-Addressing is mandatory.

Selected
WS-Addressing is mandatory. Servers return a fault if they receive a message that does not contain a WS-Addressing header. Clients always include WS-Addressing headers in SOAP messages.

Messaging style:

Specifies the messaging style supported by this policy set.

Use the radio buttons to configure the messaging style.

- Select **Synchronous and asynchronous** to specify that there is no restriction on the targeting of response messages.
- Select **Synchronous only** to specify that response messages must be targeted at the WS-Addressing anonymous URI. You might want to use this messaging style in the following situations:
  - The SOAP headers are not signed, and WS-Security is not enabled. Specifying the synchronous message exchange pattern prevents the server sending messages to a third party, thereby preventing the server from potentially taking part in a Denial of Service attack.
  - Clients with a NAT device between themselves and the endpoint. In this configuration, non-anonymous URIs cannot be routed. Use this setting to prevent the client from sending a message containing a ReplyTo endpoint reference with a non-anonymous URI.
- Select **Asynchronous only** to specify that response messages must not be targeted at the WS-Addressing anonymous URI. This setting does not mean that all non-anonymous URIs are supported, therefore a server can return a fault if it receives a response endpoint reference that it cannot process. You might want to use this messaging style if the endpoint has a very long-running invocation time, and you do not want to hold the connection open while waiting for a response.

The following table shows how the messaging style options correspond to WS-Policy assertions.

Table 7.

Messaging style	WS-Policy mapping
Synchronous and asynchronous	wsam:AnonymousResponses or wsam:NonAnonymousResponses
Synchronous only	wsam:AnonymousResponses
Asynchronous only	wsam:NonAnonymousResponses

Required No
Data type Radio button

Configuring the HTTP transport policy:

When working with policy sets in the administrative console, you can customize policies to ensure message security. You can customize the Hypertext Transfer Protocol (HTTP) transport policy configuration or use the policy as it is provided with the default settings.

Before you begin

You can configure some settings for default policies for custom policy sets. The provided default policy sets cannot be edited. To customize a policy set, you must create a copy of the default policy set or create a new policy set and specify the policies for the custom policy set.

About this task

You can configure HTTP transport with the HTTP transport policy. HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol that can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred. HTTP features and HTTP connections properties are applied to outbound messages for both the service client and service provider.

You can only configure a policy through a policy set. Therefore, before you can configure the HTTP transport policy, a policy set must exist that contains the HTTP transport policy. The provided default WSHTTPS policy set is read only and it cannot be edited. To customize a policy set that contains the HTTP transport policy, you must first create a copy of the WSHTTPS default policy set or create a new policy set and add the HTTP transport policy to the new policy set.

Note: The WSHTTPS default policy set contains the HTTP transport policy, the SSL transport policy and WS-Addressing policy. If you do not require the SSL transport policy or the WS-Addressing policy, you can customize your copy of the WSHTTPS default policy set to delete the policies that you do not require.

After you have created a copy of the WSHTTPS default policy set or created a new policy set with the HTTP transport policy added, you can customize the HTTP transport policy. Use the HTTP transport policy settings panel to customize the values of the HTTP transport policy properties such as read or write timeout values. Your customized values for the HTTP transport policy now apply for your policy set that contains that custom HTTP transport policy. You can attach this policy set containing your customized HTTP transport policy to your Java API for XML-Based Web Services (JAX-WS) application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, see the documentation for managing policy sets for service providers and service clients at the application level.

For example, if you have multiple policy sets, *mypolicyset1* and *mypolicyset2*, containing the HTTP transport policy, you can customize the HTTP transport policy for each policy set to reflect different properties, such as timeout values. Now, you can attach these customized policy sets to one or more applications and these applications will use the HTTP property values associated with the HTTP transport policy that is contained within the attached policy set.

1. Create a policy set that contains the HTTP transport policy.
   a. Create a custom policy set.

      From the administrative console, click **Services > Policy Sets > Application policy sets** . From this panel you can create a new policy set, copy an existing default policy set such as WSHTTPS, import a copy of a policy set from the default repository or you can import an existing policy set from your specified location.

b. Add the HTTP transport policy to the policy set. From the administrative console, click **Services > Policy Sets > Application policy sets > *policy_set_name***. In the policy collection, click **HTTP transport**. The HTTP transport window displays options for configuring the HTTP settings for the transport policy.

c. In the **Protocol Version** drop down list, click the HTTP version to use. HTTP 1.1 is the default setting but HTTP 1.0 is also available. Selecting HTTP 1.1 enables more of the function on the rest of the HTTP transport window as some of the options are not available for HTTP version 1.0.

d. Complete the HTTP Features section. The following check boxes determine which HTTP features are enabled for this transport:

Session Enabled
Whether the HTTP session is enabled when a message is sent.

Enable chunked transfer encoding
Whether chunked transfer encoding is enabled when a message is sent. This option is only available if HTTP 1.1 is selected in the **Protocol version** field (it is greyed out and disabled if HTTP 1.0 is selected).

Send expect ″100-request″ header
Displays whether the expect ″100-request″ header is enabled when a message is sent. This option is only available if HTTP 1.1 is selected in the **Protocol version** field (it is greyed out and disabled if HTTP 1.0 is selected).

Accept URL redirection automatically
Displays whether the URL is automatically redirected when a message is sent.

Compress request content
Displays whether the request content is compressed when a message is sent.

Compress response content
Displays whether the response content is compressed when a message is sent.

e. Complete the HTTP Connections section. The following fields determine how HTTP connections are configured for this transport:

Read timeout
Displays the length of time, in seconds, for the read to time out when a message is sent.

Write timeout
Displays the length of time, in seconds, for the write to time out when a message is sent.

Connection timeout
Displays the length of time, in seconds, for the connection to time out when a message is sent.

Use persistent connection
Displays whether a persistent connection is to be used when a message is sent. This option is only available if HTTP 1.1 is selected in the **Protocol version** field.

Resend enabled
Displays whether or not a message can be resent. Click this check box to enable a message to be sent again.

2. Customize the HTTP transport provider bindings.

a. Navigate to the HTTP transport provider bindings. From the administrative console, click **Services > Policy Sets > General provider policy set bindings > *provider_policy_set_binding_name* > HTTP transport**.

The HTTP transport (bindings) window displays options for configuring the HTTP transport bindings.

b. Specify the properties for the Proxy for outbound asynchronous service responses.

The following fields determine proxy specifications for outbound asynchronous service responses:

Host Displays the host name for the outbound asynchronous service responses proxy.

Port Displays the port number for the outbound asynchronous service responses proxy. You can enter or edit the port number.

User name
Displays the user name for the outbound asynchronous service responses proxy.

Password
Displays a placeholder for the password for the outbound asynchronous service responses proxy. You can enter or edit the password. The actual password is masked.

Confirm password
Displays a placeholder for the password for the outbound asynchronous service responses proxy that must match the one in the **Password** field. The actual password is masked.

c. Specify the properties for the Basic authentication for outbound asynchronous responses.

The following fields determine authentication specifications for outbound asynchronous responses:

User name
Displays the user name for basic authentication of outbound asynchronous responses.

Password
Displays a placeholder for the password for basic authentication of outbound asynchronous responses. The actual password is masked.

Confirm password
Displays a placeholder for the password for basic authentication of outbound asynchronous responses that must match the one in the **Password** field. The actual password is masked.

3. Customize the HTTP transport client bindings.

a. Navigate to the HTTP transport client bindings. From the administrative console, click **Services > Policy Sets > General client policy set bindings > *provider_policy_set_binding_name* > HTTP transport**.

The HTTP transport (bindings) window displays options for configuring the HTTP transport bindings.

b. Specify the properties for the Proxy for outbound service requests. The following fields determine proxy specifications for outbound service requests:

Host Displays the host name for the outbound service request proxy.

Port Displays the port number for the outbound service request proxy.

User name
Displays the user name for the outbound service request proxy.

Password
Displays a placeholder for the password for the outbound service request proxy. The actual password is masked.

Confirm password
Displays a placeholder for the password for the outbound service request proxy that must match the one in the **Password** field. The actual password is masked.

c. Specify the properties for Basic authentication for outbound service requests. The following fields determine authentication specifications for outbound service requests:

User name
Displays the user name for basic authentication of outbound service requests.

Password
Displays a placeholder for the password for basic authentication of outbound service requests. The actual password is masked.

Confirm password
Displays a placeholder for the password for basic authentication of outbound service requests that must match the one in the **Password** field. The actual password is masked.

a.

Results

After you have customized the HTTP transport policy, the associated policy set uses this policy to protect message transmission.

Example

You can attach policy sets to an application, its services, endpoints, or operations. In this example scenario, suppose you have two different JAX-WS service clients for your application, but you want to use different HTTP transport property values for each service client. Specifically, you want to configure a different read or write timeout value for each service client. To modify the HTTP timeout values, you can edit the values of the HTTP transport policy that is contained within the policy set that is attached to your application or in this case, your service client. This change affects all applications to which the policy set containing the custom HTTP transport policy is attached.

This example describes the steps for configuring different read, write, and connection timeout values for service clients deployed in the same application server. This example makes the following assumptions:

- There are two JAX-WS service clients, ServiceClient1 and ServiceClient2, that are deployed in the application server.
- The HTTP transport policy has not been previously attached to these applications.

1. Create two new policy sets and add the HTTP transport policy to them. For example: *HTTPServiceClient1Policy* and *HTTPServiceClient2Policy*
   a. Click **Services > Policy sets > Application policy sets > New** .
   b. Enter the name of the new application policy set, *HTTPServiceClient1Policy*.
   c. From the Policies collection, click **Add > HTTP transport**.
   d. Click **Apply** and **Save** to save your changes to the master configuration.
   e. Repeat these steps to create the HTTPServiceClient2Policy.
2. Customize the HTTP transport policy settings for the newly created HTTPServiceClient1Policy and HTTPServiceClient2Policy policy sets. For example, customize the read and write timeout values for the HTTP transport policy contained in the HTTPServiceClient1Policy policy set and the connection timeout value for the HTTP transport policy contained in the HTTPServiceClient2Policy policy set.
   a. Click **Services > Policy sets > Application policy sets >***HTTPServiceClient1Policy* .
   b. From the Policies collection, click **HTTP transport**.
   c. From the HTTP transport policy configuration panel, change the HTTP connection read and write timeout values to 500 seconds.
   d. Click **Apply** and **Save** to save your changes to the master configuration.
   e. Click **Services > Policy sets > Application policy sets >***HTTPServiceClient2Policy* .
   f. From the Policies collection, click **HTTP transport**.
   g. From the HTTP transport policy configuration panel, change the HTTP connection timeout value to 360 seconds.
   h. Click **Apply** and **Save** to save your changes to the master configuration.
3. Attach the custom HTTP transport policy, HTTPServiceClient1Policy, to your application, ServiceClient1. Similarly, attach the custom HTTP transport policy, HTTPServiceClient2Policy, to ServiceClient2.
   a. Click **Services > Service clients >** *ServiceClient1*.
   b. From the Policy set attachments collection, select the service, *ServiceClient1*.

c. Click **Attach Client Policy Set** and click on *HTTPServiceClient1Policy*.
d. Click **Save** to save your changes to the master configuration.
e. Click **Services > Service clients >** *ServiceClient2*.
f. From the Policy set attachments collection, select the service, *ServiceClient1*.
g. Click **Attach Client Policy Set** and click on *HTTPServiceClient2Policy*.
h. Click **Save** to save your changes to the master configuration.

As a result, the ServiceClient1 application now has the HTTPServiceClient1Policy attached and the HTTP sessions will use a read and write timeout value of 500 seconds. The ServiceClient2 application has the HTTPServiceClient2Policy attached and the HTTP sessions will use a connection timeout value of 360 seconds.

What to do next

You can customize policies to ensure message security by configuring the SSL transport policy.

HTTP transport policy settings:

Use this page to define HTTP transport policy configuration. HTTP features and HTTP connection policies are applied to outbound messages. Any changes to the HTTP transport policy from this console page affects all Java API for XML-Based Web Services (JAX-WS) applications to which this custom HTTP transport policy is attached.

To view this administrative console page, click **Services > Policy sets > Application policy sets** >*policy_set_name* > **HTTP transport**, where *policy_set_name*, applies to any policy set that contains HTTP transport policy.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

You can only configure a policy through a policy set. Therefore, before you can configure the HTTP transport policy, a policy set must exist that contains the HTTP transport policy.

The WSHTTPS default policy set is provided with the application server and it contains the HTTP transport policy, the SSL transport policy and the WS-Addressing policy. The provided default WSHTTPS policy set is read only and it cannot be edited. To customize a policy set that contains the HTTP transport policy, you must first create a copy of the WSHTTPS default policy set or create a new policy set and add the HTTP transport policy to the new policy set.

After you customize values for the HTTP transport policy, these values now apply for your policy set that contains that custom HTTP transport policy. You can attach this policy set that contains your customized HTTP transport policy to your application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, see the documentation for managing policy sets for service providers and service clients at the application level.

Protocol version:

Specifies the version of HTTP protocol to use. Use this list to specify the version of HTTP protocol. The default value is `HTTP 1.1`. The `HTTP 1.0` value is also a valid option.

Some of the remaining options on the HTTP Transport panel only work with HTTP Version 1.1. The following brief descriptions compare these options:

HTTP 1.0
Allows messages to be in MIME-like format, containing meta information about the data

transferred and modifiers on the request, response, or both. However, HTTP 1.0 does not sufficiently address the effects of hierarchical proxies, caching, the need for persistent connections, or virtual hosts.

HTTP 1.1
Enables each of two communicating applications to determine the true capabilities or the other. This protocol includes more stringent requirements than HTTP 1.0 to ensure reliable implementation of features.

Session enabled:

Specifies whether the HTTP session is enabled when a message is sent. Select this check box to enable an HTTP session.

If this property is used within a policy set that is attached to a service client, then it indicates whether HTTP session information is propagated to subsequent requests invoked by the same client application. If the property is enabled, the HTTP session information is returned to the service client in a response message is sent in subsequent requests invoked using the same RequestContext object.

If this property is used within a policy set that is attached to a service provider, then it indicates whether or not a new HTTP session is created when a request is being processed. If the property is enabled, then as a request is being processed, a new HTTP session is created if one does not already exist. This HTTP session information is then returned to the service client in the response message.

Enable chunked transfer encoding:

Specifies whether chunked transfer encoding is enabled when a message is sent. Select this check box to enable a chunked transfer encoding. This option is only available if you select HTTP 1.1 in the **Protocol version** field. This option is disabled if you selected the HTTP 1.0 protocol.

The default for this property is `true`.

Send expect ″100-request″ header:

Specifies whether the `expect "100-request"` header is enabled when a message is sent. Select this check box to enable the `expect "100-request"` header. This option is only available if you selected HTTP 1.1 in the **Protocol version** field. This option is disabled if you selected the HTTP 1.0 protocol.

The purpose of the 100 status is to allow a client that is sending a request message with a request body to determine if the origin server accepts the request, based on the request headers, before the client sends the request body. In some cases, you might not want the client to send the body if the server rejects the message without looking at the body.

The Expect request-header field is used to indicate that particular server behaviors are required by the client. A server that cannot comply with any of the expectation values in the Expect field if a request responds with an appropriate error status.

Accept URL redirection automatically:

Specifies whether the automatic URL redirection is accepted when a message is sent. Select this check box to enable a URL that has been automatically redirected to be accepted.

Compress request content:

Specifies whether the request content is compressed when a message is sent. Content coding is used to allow a document to be compressed without losing the identity of its underlying media type and without loss of information. Select this check box to enable request content that you want to compress. Clicking

the Compress request content button enables the **Compression format** option to select the compression method. The default value for the compression format is `gzip`.

Compress response content:

Specifies whether the response content is compressed when a message is sent. Content coding is used to allow a document to be compressed without losing the identity of its underlying media type and without loss of information. Select this check box to enable response content that you want to compress. Clicking the Compress response content button enables the **Compression format** option as he compression method. The default value for the compression format is `gzip`.

Read timeout:

Specifies the length of time, in seconds, for the Web services client to completely read the SOAP response. If the read process does not complete within the specified time, a SOAP fault error is generated on the client machine.

Write timeout:

Specifies the length of time, in seconds, for the write action to time out when a message is sent. Specify the time, in seconds, to enable the write to time out length of time.

Connection timeout:

Specifies the length of time, in seconds, for the connection to time out when a message is sent. Specify the time, in seconds, to enable the connection to time out length of time.

Use persistent connection:

Specifies whether a persistent connection is used when a message is sent. Select this check box to enable use of a persistent connection. This option is only available if you selected HTTP 1.1 in the **Protocol version** field. This option is disabled if you selected the HTTP 1.0 protocol.

Resend enabled:

Specifies whether a message can be resent. Select this check box to resend a message.

HTTP transport bindings settings:

Use this page to define the HTTP transport bindings for the HTTP transport policy.

To configure the HTTP transport bindings for the HTTP transport policy, perform the following:

1. Navigate to the general bindings collection panel using either the **Services > Policy sets > General client policy set bindings** or **Services > Policy sets > General provider policy set bindings** path.
2. Click a general binding in the Name column.
3. Click the **HTTP transport** policy in the Policies table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Note: You can also configure HTTP transport properties such as read or write timeout values for JAX-WS applications that are deployed in the same application server. If you want to customize these HTTP properties, you must edit the HTTP transport policy. To customize the HTTP transport policy settings, click **Services > Policy sets > Application policy sets >*policy_set_name*>HTTP transport policy** where *policy_set_name* applies to any policy set that contains the HTTP transport policy. Your customized values for the HTTP transport policy now apply for your policy set that contains that custom HTTP transport policy. You can attach this policy set that contains your

customized HTTP transport policy to your application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, see the documentation for managing policy sets for service providers and service clients at the application level.

Proxy for outbound service requests – Host:

Specifies the host name for the outbound service request proxy. In this field, you can change the name of the host if you are editing an existing set of application specific bindings or you can enter the host name for the outbound service request proxy.

Proxy for outbound service requests – Port:

Specifies the port number for the outbound service request proxy. You can edit an existing port number, or enter a new port number for the outbound service request proxy in this field.

Proxy for outbound service requests – User name:

Specifies the user name for the outbound service request proxy. Enter a user name in this field.

Proxy for outbound service requests – Password:

Specifies a placeholder for the outbound service request proxy password. The actual password is masked.

Proxy for outbound service requests – Confirm password:

Specifies a placeholder for the outbound service request proxy password. Re-enter the password you entered in the **Password** field. The actual password is masked.

Basic authentication for outbound service requests – User name:

Specifies the user name for basic authentication of outbound service requests. Enter or edit the user name.

Basic authentication for outbound service requests – Password:

Specifies a placeholder for basic authentication of outbound service requests password. The actual password is masked.

Basic authentication for outbound service requests – Confirm password:

Specifies a placeholder for basic authentication of outbound service requests password. Re-enter the same password as in the **Password** field. The actual password is masked.

Proxy for outbound asynchronous service responses – Host:

Specifies the host name for the outbound asynchronous service responses proxy. You can enter or edit the host name.

Proxy for outbound asynchronous service responses – Port:

Specifies the port number for the outbound asynchronous service responses proxy. You can enter or edit the port number.

Proxy for outbound asynchronous service responses – User name:

Specifies the user name for the outbound asynchronous service responses proxy. You can enter or edit the user name.

Proxy for outbound asynchronous service responses – Password:

Specifies a placeholder for the outbound asynchronous service responses proxy password. You can enter or edit the password. The actual password is masked.

Proxy for outbound asynchronous service responses – Confirm password:

Specifies a placeholder for the outbound asynchronous service responses proxy password. You can re-enter or edit the password. The actual password is masked.

Basic authentication for outbound asynchronous service responses – User name:

Specifies the user name for basic authentication of outbound asynchronous responses. You can enter or edit the user name.

Basic authentication for outbound asynchronous service responses – Password:

Specifies a placeholder for basic authentication of outbound asynchronous responses password. You can enter or edit the password in this field. The actual password is masked.

Basic authentication for outbound asynchronous service responses – Confirm password:

Specifies a placeholder for basic authentication of outbound asynchronous responses password. Re-enter the password in this field. The actual password is masked.

Custom Properties – Name:

Specifies the name of custom property. Custom properties are not initially displayed in this column until you define them.

Custom Properties – Value:

Specifies the value of the custom property. With the **Value** entry field, you can enter, edit, or delete the value for a custom property.

Click one of the following buttons to enable the action described:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Delete	Removes the selected custom property.
Edit	You can edit a selected custom property. It is only displayed when one or more properties exist.

Configuring the Java Message Service (JMS) transport policy:

You can define a Java Message Service (JMS) transport policy configuration if you are using SOAP over JMS with your Java API for XML-Based Web Services (JAX-WS) applications.

Before you begin

You can configure some settings for policies for custom policy sets. The provided default policy sets cannot be edited. You must create a copy of the default policy set or create a new policy set in order to specify the policies for it.

About this task

When using the SOAP over JMS transport with JAX-WS applications, you can customize the transport by configuring the JMS transport policy. The SOAP over JMS transport provides an alternative to HTTPS for transporting SOAP requests and response messages between clients and servers. See the documentation on using SOAP over JMS to transport Web services to learn more about this transport protocol.

You can only configure a policy through a policy set. Therefore, before you can configure the JMS transport policy, a policy set must exist that contains the JMS transport policy. To customize a policy set that contains the JMS transport policy, you must first create a policy set and add the JMS transport policy to the new policy set.

Use the JMS transport policy settings panel to customize the values of the JMS transport policy properties, such as the request timeout value. Your customized values for the JMS transport policy now apply for your policy set that contains that custom JMS transport policy. You can attach this policy set containing your customized JMS transport policy to your JAX-WS application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, see the documentation for managing policy sets for service providers and service clients at the application level.

1. Create a policy set that contains the JMS transport policy.
   a. Create a custom policy set. From the administrative console, click **Services > Policy Sets > Application policy sets**. From this panel you can create a new policy set, import a copy of a policy set from the default repository, or you can import an existing policy set from your specified location.
   b. Add the JMS transport policy to the policy set. From the administrative console, click **Services > Policy Sets > Application policy sets > *policy_set_name***. In the policy collection, click **JMS transport**. The JMS transport window displays options for configuring the JMS settings for the transport policy.
   c. Specify the JMS connection properties for the JMS transport requests. The following fields configure the JMS features for this transport:

      Request timeout
      Specifies whether to enable a request timeout value. The request timeout value is the amount of time that the client waits for a response after sending the request to the server. The range is from 0 to 2147483647.

      Allow transactional messaging for one-way and asynchronous operations
      Specifies to enable a client to use transactions in one-way or asynchronous two-way requests. Select this check box to enable transactional messaging.

      When this option is selected, the client runtime environment exchanges SOAP request and response messages with the server over the JMS transport in a transactional manner if the client is operating under a transaction. This process indicates that the clients transaction is used to send the SOAP request message to the destination queue or topic, and the server receives the request message only after the client commits the transaction. Similarly, the server receives the request message under the control of a container-managed transaction and sends the reply message, if applicable, back to the client using that same transaction. The client then receives the reply message only after the server transaction has been committed.

If this option is not selected, then the client and server runtime environments perform messaging operations in a non-transactional manner as transactions are temporarily suspended for the JMS request. The transactions are enabled again after the request has completed.

Note: Transactional messaging operations are not supported for two-way synchronous operations as this leads to a deadlock condition.

2. Customize the JMS transport provider bindings.
   a. Navigate to the JMS transport provider bindings. From the administrative console, click **Services > Policy Sets > General provider policy set bindings > *provider_policy_set_binding_name* > JMS transport**.

      The JMS transport provider bindings window displays options for defining basic authentication for asynchronous service responses and custom properties for the JMS service provider binding configuration.

   b. Specify the properties for basic authentication for asynchronous service responses.

      You can use the JMS transport provider policy bindings to configure a service that uses the JMS transport to send asynchronous response messages back to the client. The application server runtime environment uses the user name and password that you configure when connecting to the JMS messaging provider and this configuration enables the service to send an asynchronous response message to the client in a secure manner.

      The following fields determine the authentication requirements for responses from the server:

      User name
      Specifies the user name for the asynchronous service responses for the service provider.

      Password
      Specifies a placeholder for the password for the asynchronous service responses from the service provider. You can enter or edit the password in this field. The actual password is masked.

      Confirm password
      Specifies a placeholder for the password for the asynchronous service responses from the service provider that must match the one in the **Password** field. The actual password is masked.

   a.

3. Customize the JMS transport client bindings.
   a. Navigate to the JMS transport client bindings. From the administrative console, click **Services > Policy Sets > General client policy set bindings > *client_policy_set_binding_name* > JMS transport**.

      The JMS transport provider bindings window displays options for defining basic authentication for outbound service requests and custom properties for the JMS client binding configuration.

   b. Specify the Basic Authentication for Outbound Service Requests properties.

      You can use the JMS transport client policy bindings to configure a client that uses the JMS transport to send a request message to the server. The client runtime environment uses the user name and password that you configure when connecting to the JMS messaging provide. This configuration enables the client to send the request message to the server in a secure manner.

      The following fields determine the authentication requirements for requests sent to the server:

      User name
      Specifies the user name that is used by the client runtime when connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. Enter a user name in this field.

      Password
      Specifies a placeholder for the password that is used by the client runtime when

connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. You can enter or edit the password in this field. The actual password is masked.

Confirm password
Specifies a placeholder for the password that is used by the client runtime when connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. Re-enter the password in this field. This password must match the one in the **Password** field. The actual password is masked.

Results

After you have customized the JMS transport policy, the associated policy set uses this policy to configure the runtime behavior of the SOAP over JMS transport.

Example

You can attach policy sets to an application, its services, endpoints, or operations. In this example scenario, suppose you have two different JAX-WS service clients for your application, but you want to use different JMS transport request timeout values for each service client. To modify the JMS request timeout values, you can edit the values of the JMS transport policy that is contained within the policy set that is attached to your application or in this case, your service client. This change affects all applications to which the policy set containing the custom JMS transport policy is attached.

This example describes the steps for configuring different request timeout values for service clients deployed in the same application server. This example includes the following assumptions:

- Two JAX-WS service clients exist, ServiceClient1 and ServiceClient2, that are deployed in the application server.
- The JMS transport policy has not been previously attached to these applications.

1. Create two new policy sets and add the JMS transport policy to them. For example: *JMSServiceClient1Policy* and *JMSServiceClient2Policy*
   a. Click **Services > Policy sets > Application policy sets > New** .
   b. Enter the name of the new application policy set, `JMSServiceClient1Policy`.
   c. From the Policies collection, click **Add > JMS transport**.
   d. Click **Apply** and **Save** to save your changes to the master configuration.
   e. Repeat these steps to create the JMSServiceClient2Policy.
2. Customize the JMS transport policy settings for the newly created JMSServiceClient1Policy and JMSServiceClient2Policy policy sets. For example, set the request timeout value to `180` seconds for the JMS transport policy contained in the JMSServiceClient1Policy. The JMS transport policy contained in the JMSServiceClient2Policy specifies `300` seconds as the request timeout value.
   a. Click **Services > Policy sets > Application policy sets > *JMSServiceClient1Policy*** .
   b. From the Policies collection, click **JMS transport**.
   c. From the JMS transport policy configuration panel, specify `180` seconds for the request timeout value.
   d. Click **Apply** and **Save** to save your changes to the master configuration.
   e. Click **Services > Policy sets > Application policy sets > *JMSServiceClient2Policy*** .
   f. From the Policies collection, click **JMS transport**.
   g. From the JMS transport policy configuration panel, specify 300 seconds for the request timeout value.
   h. Click **Apply** and **Save** to save your changes to the master configuration.
3. Attach the custom JMS transport policy, JMSServiceClient1Policy, to your application, ServiceClient1. Similarly, attach the custom JMS transport policy, JMSServiceClient2Policy, to ServiceClient2.

a. Click **Services > Service clients >** *ServiceClient1*.
b. From the Policy set attachments collection, select the service, *ServiceClient1*.
c. Click **Attach Client Policy Set**, and click *JMSServiceClient1Policy*.
d. Click **Save** to save your changes to the master configuration.
e. Click **Services > Service clients >** *ServiceClient2*.
f. From the Policy set attachments collection, select the service, *ServiceClient1*.
g. Click **Attach Client Policy Set**, and click *JMSServiceClient2Policy*.
h. Click **Save** to save your changes to the master configuration.

As a result, the ServiceClient1 application now has the JMSServiceClient1Policy attached, and the JMS sessions use a request timeout of 180 seconds. The ServiceClient2 application has the policy, JMSServiceClient2Policy, attached and the JMS sessions use a request timeout of 300 seconds.

What to do next

You can customize other policies that you might need for your application.

JMS transport policy settings:

Use this page to configure settings for the Java Message Service (JMS) transport policy. You can configure a client that is using the JMS transport policy to exchange request and response messages with the server.

To view this administrative console page:
1. Click **Services > Policy Sets > Application policy sets**
2. Click ***policy_set_name***
3. In the policy collection, click **JMS transport**

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

You can only configure a policy through a policy set. Therefore, before you can configure the JMS transport policy, a policy set must exist that contains the JMS transport policy.

To customize a policy set that contains the JMS transport policy, you must create a new policy set, import a copy of a policy set from the default repository, or you can import an existing policy set from your specified location. After you have an editable policy set, you can add the JMS transport policy to your policy set.

After you customize values for the JMS transport policy, these values now apply for your policy set that contains that customized JMS transport policy. You can attach this policy set that contains your customized JMS transport policy to your application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, read about managing policy sets for service providers and service clients at the application level.

Client JMS connection properties - Request timeout:

Specifies the request timeout value. The request timeout value is the amount of time that the client waits for a response after sending the request to the server. The default value is 300 seconds.

Client JMS connection properties - Allow transactional messaging for one-way and asynchronous operations:

Specifies to enable a client to use transactions in one-way or asynchronous two-way requests. Select this check box to enable transactional messaging.

If this option is selected, the client runtime exchanges SOAP request and response messages with the server over the JMS transport in a transactional manner if the client is operating under a transaction. Therefore, the client transaction is used to send the SOAP request message to the destination queue or topic, and the server receives the request message only after the client commits the transaction. Similarly, the server receives the request message under the control of a container-managed transaction and sends the reply message, if applicable, back to the client using that same transaction. The client then receives the reply message only after the server transaction is committed.

If this option is not selected, the client and server runtimes perform messaging operations in a non-transactional manner.

Note: Transactional messaging operations are not supported for two-way synchronous operations as this leads to a deadlock condition.

.

JMS transport bindings:

Use this page to define the Java Message Service (JMS) transport provider or client bindings configuration.

If you are using JMS transport provider bindings, to view this administrative console page, complete the following actions:

1. Click **Services > Policy Sets > General provider policy set bindings**.
2. Click ***provider_policy_set_binding_name***.
3. In the policy collection, click **JMS transport**.

If the JMS transport policy has not been added to the selected general provider policy set, then use the create general provider bindings panel to add the JMS transport policy to the selected general provider policy set.

You can use the JMS transport provider policy bindings to configure a service that uses the JMS transport to send asynchronous response messages back to the client. The application server run time uses the user name and password that you configure when connecting to the JMS messaging provider and this configuration enables the service to send an asynchronous response message to the client in a secure manner.

If you are using JMS transport client bindings, to view this administrative console page, complete the following actions:

1. Click **Services > Policy Sets > General client policy set bindings**.
2. Click ***client_policy_set_binding_name***.
3. In the policy collection, click **JMS transport**.

If the JMS transport policy has not been added to the selected general client policy set, then use the create general client bindings panel to add the JMS transport policy to the selected general client policy set.

You can use the JMS transport client policy bindings to configure a client that uses the JMS transport to send a request message to the server. The client run time uses the user name and password that you specify when connecting to the JMS messaging provider, and this configuration enables the client to send the request message to the server in a secure manner.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Note: You can also configure JMS transport properties, such as request timeout values or whether to enable transactional messaging for one-way asynchronous operations for JAX-WS applications that are deployed in the same application server. If you want to customize these JMS properties, you must edit the JMS transport policy. To customize the JMS transport policy settings, click **Services > Policy sets > Application policy sets > *policy_set_name* >JMS transport policy**, where *policy_set_name* applies to any policy set that contains the JMS transport policy. Your customized values for the JMS transport policy now apply for your policy set that contains that custom JMS transport policy. You can attach this policy set that contains your customized JMS transport policy to your application, its services, endpoints, or operations. This change affects all JAX-WS applications to which that policy set is attached. To learn more about attaching policy sets to applications, see the documentation for managing policy sets for service providers and service clients at the application level.

Related tasks

"Modifying policies using the administrative console" on page 905
With your policy sets, you can define policies for WS-Addressing, WS-Security, WS-ReliableMessaging, WS-Transaction, HTTP transport, Java Messaging Service (JMS) transport, and Secure Sockets Layer (SSL) transport. The policies for all but WS-Security are relatively straightforward to define.

"Defining and managing policy set bindings" on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

"Managing policy sets and bindings for service providers at the application level using the administrative console" on page 743
Use this administrative console task to manage policy sets for an application or its services, endpoints, and operations.

"Managing policy sets and bindings for service clients at the application level using the administrative console" on page 760
Use this administrative console task to manage policy sets for service clients applications or its services, endpoints, or operations.

Using SOAP over Java Message Service to transport Web services
You can use the SOAP over Java Message Service (JMS) transport protocol as an alternative to SOAP over HTTP for communicating SOAP messages between clients and servers.

Related reference

"JMS transport policy settings" on page 925
Use this page to configure settings for the Java Message Service (JMS) transport policy. You can configure a client that is using the JMS transport policy to exchange request and response messages with the server.

Basic Authentication – User name:

For the service provider, this field specifies the user name for the asynchronous service responses. For the client, this field specifies the user name that is used by the client runtime when connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. Enter a user name in this field.

Basic Authentication – Password:

For the service provider, this field specifies a placeholder for the password of the asynchronous service responses. For the client, this field specifies the password that is used by the client runtime when connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. You can enter or edit the password in this field. The actual password is masked.

Basic Authentication – Confirm password:

For the service provider, this field specifies a placeholder for the password for the asynchronous service responses. For the client, this field specifies the password that is used by the client runtime when connecting to the JMS messaging provider to send an outbound request to the destination queue or topic. Re-enter the password in this field. The actual password is masked.

Custom Properties – Name:

Specifies the name of custom property. Custom properties are not initially displayed in this column until you define them.

Click one of the following buttons to enable the described action:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Delete	Removes the selected custom property.
Edit	Edit a selected custom property. This button is only displayed when one or more properties exist.

Custom Properties – Value:

Specifies the value of the custom property. With the **Value** entry field, you can enter, edit, or delete the value for a custom property.

Configuring the SSL transport policy:

When working with policy sets in the administrative console, you can customize policies to ensure message security by configuring the SSL transport policy.

Before you begin

The default policy sets provided with the product cannot be edited. To configure custom policy sets, you must first copy the default policy set or create a completely new policy set in order to specify the policies for it. See creating policy sets using the administrative console.

About this task

The SSL transport policy provides the SSL transport security for the Hypertext Transfer Protocol (HTTP) protocol with Web services applications. To view the default SSL transport policy set with the SSL transport policy, click **Services > Policy sets > Application policy sets > WSHTTPS default > SSL transport**.

1. To edit the SSL transport policy, click a policy set that you have created or customized from the default. Select the SSL transport policy applicable check boxes to enable the SSL functions. The following check boxes determine how SSL security is configured for this transport:
   - **Enable for outbound service requests**

     Displays whether the SSL security transport is enabled for outbound service requests.
   - **Enable for outbound asynchronous service responses**

     Displays whether the SSL security transport is enabled for outbound asynchronous service responses.
   - **Enable for inbound service responses**

     Displays whether the SSL security transport is enabled for inbound service responses.
2. To configure the binding for the SSL transport policy, click **Services > Policy sets > General client policy set bindings** > *binding_name* > **SSL transport** or **Services > Policy sets > General provider policy set bindings** > *binding_name* > **SSL transport**. Select the setting to configure the SSL bindings. The SSL transport window displays options for configuring the SSL security bindings.

a. Select the setting to configure the SSL bindings for the **Outbound service requests**.
   - **SSL settings**

     Specifies the SSL security transport binding that is enabled for outbound service requests. The default value for this field is **CellDefaultSSLSettings**.
   - **SSL properties file path**

     Specifies the path of the SSL properties file that is enabled for asynchronous service responses. Enter the location of the SSL properties file to enable for asynchronous service responses.

b. Select the setting to configure the SSL bindings for the **Inbound service responses**.
   - **SSL settings**

     Specifies the SSL security transport binding that is enabled for inbound service responses. The default value for this field is **CellDefaultSSLSettings**.
   - **SSL properties file path**

     Specifies the path of the SSL properties file that is enabled for inbound service responses. Enter the location of the SSL properties file to enable for inbound service responses.

c. Select the setting to configure the SSL bindings for the **Outbound asynchronous service responses**.
   - **SSL settings**

     Specifies the SSL security transport binding that is enabled for asynchronous service responses. The default value for this field is **CellDefaultSSLSettings**.
   - **SSL properties file path**

     Specifies the file path of the SSL properties file that is enabled for outbound service requests. Enter the location of the SSL properties file to enable for outbound service requests.

   Custom properties

   Click one of the following buttons to enable the action described:

Button	Resulting Action
New	Creates a new custom property entry. To add a custom property, enter the name and value.
Delete	Removes the selected custom property.
Edit	Enables you to edit a selected custom property. It is only displayed when one or more properties exist.

Results

Once you have customized the SSL transport policy, the associated policy set uses this policy to protect message transmission. Similarly, you can also configure HTTP transport with the HTTP transport policy. Read about configuring the HTTP transport policy to learn how to configure the HTTP transport with the HTTP transport policy.

What to do next

Depending on how you are using policies, you might want to configure the HTTP transport policy or the SSL transport security bindings.

SSL transport security policy settings:

Use this page to define the secure sockets layer (SSL) transport policy configuration for policy sets.

To configure the SSL transport security for a policy set, click **Services > Policy sets > Application policy sets** > *policy_set_name* > **SSL transport**, where *policy_set_name*, applies to any policy set that contains SSL transport security. An example of such SSL transport security is WSHTTPS default.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Enable for outbound service requests:

Specifies whether the SSL security transport is enabled for outbound service requests when the client sends out requests.

Enable for outbound asynchronous service responses:

Specifies whether the SSL security transport is enabled for outbound asynchronous service responses when the service or server sends back the response.

Enable for inbound service responses:

Specifies whether the SSL security transport is enabled for inbound service responses when the client receives responses.

SSL transport security settings:

Use this page to define the secure sockets layer (SSL) transport policy binding configuration.

1. Navigate to the general bindings collection page by clicking either **Services > Policy sets > General client policy set bindings** or **Services > Policy sets > General provider policy set bindings** path.
2. Click a general binding in the Name column.
3. Click the **SSL transport** policy in the Policies table.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Outbound service requests – SSL settings:

Specifies the SSL security transport binding that is enabled for outbound service requests when the client sends out requests. The default value for this field is `CellDefaultSSLSettings`.

Outbound service requests – SSL properties file path:

Specifies the file path of the SSL properties file that is enabled for outbound service requests. Enter the location of the SSL properties file to enable for outbound service requests.

Inbound service responses – SSL settings:

Specifies the SSL security transport binding that is enabled for inbound service responses when the client receives responses. The default value for this field is `CellDefaultSSLSettings`.

Inbound service responses – SSL properties file path:

Specifies the SSL security transport binding that is enabled for inbound service responses. Enter the location of the SSL properties file to enable for inbound service responses.

Outbound asynchronous service responses – SSL settings:

Specifies the SSL security transport binding that is enabled for asynchronous service responses when the service or server sends back the response. The default value for this field is `CellDefaultSSLSettings`.

Outbound asynchronous service responses – SSL properties file path:

Specifies the path of the SSL properties file that is enabled for asynchronous service responses. Enter the location of the SSL properties file to enable for asynchronous service responses.

Outbound asynchronous service responses – Custom properties:

Specifies the name and value pair that you define for the outbound asynchronous service responses. Click **New** to add a new custom property, or click **Delete** to delete an existing SSL custom property.

Configuring the WS-Transaction policy:

When you work with policy sets in the administrative console, you can configure the WS-Transaction policy type for the WS-AtomicTransaction (WS-AT) and the WS-BusinessActivity (WS-BA) protocols. You can configure whether a client propagates, and a server receives, a WS-AT context, and whether a client propagates, and a server receives, a WS-BA context.

Before you begin

You must be working with a policy set that includes the WS-Transaction policy type.

Do not edit the policies associated with the provided default policy sets. If you need to modify the WS-Transaction policy settings, use a copy of a default policy set or create a new policy set.

About this task

You can configure the policies for the WS-AtomicTransaction and WS-BusinessActivity protocols. The WS-AT protocol supports coordination of activities so that either all the activities occur, or none of them occur. The WS-BA protocol supports coordination of activities that are more loosely coupled than atomic transactions and that therefore require a compensation process if a failure occurs in the business activity.

When you add a WS-Transaction policy, it is equivalent to setting the following deployment descriptors that are associated with an EJB or Web module:

- Use Web Services Atomic Transaction
- Send Web Services Atomic Transaction on requests
- Execute using Web Services Atomic Transaction on incoming requests

A WS-BA context is sent if the client is running in a BusinessActivity scope (BAScope). A provider runs in a BAScope if it receives a message that contains a WS-BA context, as long as the provider is set to run Enterprise JavaBeans (EJB) methods in a Business Activity scope.

1. In the navigation pane of the administrative console, click **Services** → **Policy sets** → **Application policy sets** → ***policy_set_name*** → **[Policies] WS-Transaction**. The WS-Transactions settings pane is displayed.
2. In the WS-AtomicTransaction section, select the option you require:
   - **Mandatory**. For a client, the client always propagates a WS-AT context on an outbound request. For a server, any request that is received must include a WS-AT context, otherwise the request is rejected.
   - **Supports**. For a client, the client can propagate a WS-AT context on an outbound request when that context is available. For a server, if a request includes a WS-AT context, the context is imported and established on the thread before the request is processed.
   - **Never**. For a client, the client never propagates a WS-AT context on an outbound request. For a server, any request that is received must not include a WS-AT context, otherwise the request is rejected.
3. In the WS-BusinessActivity section, select the option you require:
   - **Mandatory**. For a client, the client always propagates a WS-BA context on an outbound request. For a server, any request that is received must include a WS-BA context, otherwise the request is rejected.

- **Supports**. For a client, the client can propagate a WS-BA context on an outbound request when that context is available. For a server, if a request includes a WS-BA context, the context is imported and established on the thread before the request is processed.
- **Never**. For a client, the client never propagates a WS-BA context on an outbound request. For a server, any request that is received must not include a WS-BA context, otherwise the request is rejected.

4. Click **OK**.
5. Save your changes to the master configuration.

Results

After you configure the WS-Transaction policy, the associated policy set uses this policy to support WS-AtomicTransaction and WS-BusinessActivity.

WS-Transaction policy settings:

Use this page to specify the policies for the WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA) protocols. WS-AT supports coordination of activities so that either all the activities occur, or none of them occur. WS-BA supports coordination of activities that are more loosely coupled than atomic transactions, and that therefore, require a compensation process if an error occurs.

To view this pane in the console, click the following path: **Services** → **Policy sets** → **Application policy sets** → ***policy_set_name*** → **[Policy] WS-Transaction**, when the policy set includes the WS-Transaction policy type.

You can configure the WS-Transaction policy type for both client and provider policy sets.

WS-AtomicTransaction: Specifies behavior with the WS-AT policy. The options are:

Mandatory
For a client, the client always propagates a WS-AT context on an outbound request. If there is no transaction on the thread when the request is made, the attempt to make the request fails.

For a server, any request that is received must include a WS-AT context, otherwise the request is rejected. If any Web Services Description Language (WSDL) is generated for the Web service with which the policy type is associated, a policy assertion is included that indicates that an operation must be invoked with an atomic transaction context.

Supports
For a client, the client can propagate a WS-AT context on an outbound request when it is available. For example, a transaction is associated with the thread that makes the request, and the policy of the provider requires WS-AT context.

For a server, if a request includes a WS-AT context, the context is imported and established on the thread before the request is processed. If a request does not include a WS-AT context, the request is processed as normal. If any WSDL is generated for the Web service with which the policy type is associated, a policy assertion is included that indicates that an operation supports invocation with an atomic transaction context when that context is available.

Never For a client, the client never propagates a WS-AT context on an outbound request.

For a server, any request that is received must not include a WS-AT context, otherwise the request is rejected with a MustUnderstand error. If any WSDL is generated for the Web service with which the policy type is associated, that WSDL does not include a policy assertion for an atomic transaction context.

WS-BuinessActivity: Specifies behavior with the WS-BA policy. The options are:

Mandatory

For a client, the client always propagates a WS-BA context on an outbound request. If there is no business activity scope on the thread when the request is made, the attempt to make the request fails.

For a server, any request that is received must include a WS-BA context, otherwise the request is rejected. If any WSDL is generated for the Web service with which the policy type is associated, a policy assertion is included that indicates that an operation must be invoked with a business activity context.

Supports

For a client, the client can propagate a WS-BA context on an outbound request when it is available. For example, a business activity scope is associated with the thread that makes the request, and the policy of the provider requires a WS-BA context.

For a server, if a request includes a WS-BA context, the context is imported and established on the thread before the request is processed. If a request does not include a WS-BA context, the request is processed as normal. If any WSDL is generated for the Web service with which the policy type is associated, a policy assertion is included that indicates that an operation supports invocation with a business activity context when that context is available.

Never For a client, the client never propagates a WS-BA context on an outbound request.

For a server, any request that is received must not include a WS-BA context, otherwise the request is rejected with a MustUnderstand error. If any WSDL is generated for the Web service with which the policy type is associated, that WSDL does not include a policy assertion for a business activity context.

Configuring the WS-Security policy:

When working with policy sets in the administrative console, you can customize policies to ensure message security. The WS-Security policy can be configured to apply a message security (WS-Security) profile to requests. Message security policies are applied to requests and enforced on responses to support interoperability.

Before you begin

You can configure some settings for default policies for custom policy sets. The provided default policy sets cannot be edited. You must create a copy of the default policy set or create a completely new policy set in order to specify the policies for it.

About this task

Message security policies are applied to requests and enforced on responses to support interoperability.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

1. Use the WS-Security policy panel to begin configuring the WS-Security policy. To access the WS-Security policy panel, from the administrative console, click **Services > Policy sets > Application policy sets > *policy_set_name* > WS-Security policy**.
2. Choose which type of message security to configure.
   - Click the Main policy link to specify how message security policies are applied to requests and enforced on responses to support interoperability.
   - Click the Bootstrap policy link to configure how secure conversations are established. A bootstrap policy might already be configured. If no bootstrap policy is currently configured, first ensure that you have enabled message security with symmetric signature and encryption policies and secure conversation tokens for both integrity and confidentiality protection.

3. Use the Main policy settings panel or the Bootstrap policy settings panel to specify how message security policies are applied to requests and enforced on responses. Assertions for WS-Security versions are already generated based on assertions in the policy set. If the policy set includes a WS-S 1.1 assertion, then WS-S 1.1 itself is asserted. Configure the settings on this panel to configure main or bootstrap policy settings:
   a. Select whether Message level protection is required. Select this check box if any of the message parts should be digitally signed or encrypted or if a timestamp should be inserted in the message. It this box is unchecked, the Signature confirmation, Key symmetry, and Timestamp and Security header layout options are disabled.
   b. Specify whether signature confirmation is required. Click this check box to require signature confirmation.
   c. Configure the settings in the Key Symmetry® section. The following fields can be configured in the Key symmetry section:

      Use symmetric tokens
      Click this radio button to use symmetric tokens. You can then configure symmetric tokens with the **Symmetric signature and encryption policies** link. Click this link to access the Symmetric Signature and Encryption Policies panel where you can create the trust context in which to use symmetric tokens. Using the same token for signing and validating messages and encrypting and decrypting messages provides better performance than can be achieved with asymmetric tokens. Symmetric tokens should be used within a trust context.

      Use asymmetric tokens
      Click this link to access the Asymmetric Signature and Encryption Policies panel where you can create the trust context (message integrity and confidentiality) in which to use asymmetric tokens. You can do this by specifying which token type to use for the initiator and recipient signature as well as the initiator and recipient encryption.

      Include timestamp in header
      Click this check box to include a timestamp in the header. You can then specify if the timestamp is positioned first or last in the header by using the Security header layout radio button options:
      - **Strict: Declarations must precede use**
      - **Layout (Lax): Order of contents can vary**
      - **Lax but timestamp required first in header**
      - **Lax but timestamp required last in header**

   d. Optional: Click the **Algorithms** link under the **Policy Details** section if you want to access the Algorithms panel to view and select from available algorithms. The available algorithms include cryptographic algorithms and their key lengths, as well as canonicalization algorithms for reconciling XML differences. Click this link to view the cryptographic and cannonicalization algorithms that are supported.
   e. Optional: Configure the request settings. Click either of the following links to configure request settings:

      Request message part protection
      Links to configuration for request message part protection. Click this link to define which message parts are to be protected and how that protection is provided.

      Request token policies
      Links to configuration for request token policies. Click this link to define policies that specify which types of security tokens are supported and the properties of those token types.

   f. Optional: Configure the response settings. Click either of the following links to configure response settings:

Response message part protection
Links to configuration for response message part protection. Click this link to define which message parts are to be protected and how that protection is provided.

Response token policies
Links to configuration for response token policies. Click this link to define policies that specify which types of security tokens are supported and the properties of those token types.

Results

Once you have customized the WS-Security policy, the associated policy set uses this policy to protect messages.

WS-Security policy settings:

Use this page to configure the WS-Security policy and apply a message security WS-Security profile to requests. WS-Security policies are applied to requests and enforced on responses to support inter-operability.

To view this administrative console page, click **Services > Policy sets > Application policy sets > *policy_set_name* > WS-Security policy**.

Main policy:

Links to configuration settings for a main policy. Use the Main policy link if you want to configure how WS-Security policies are applied to requests and enforced on responses to support interoperability.

Secure conversation bootstrap policy:

Links to configuration settings for a secure conversation bootstrap policy. Use the bootstrap policy link if you want to configure how secure conversations are established. To configure a secure conversation bootstrap policy, first ensure that you have enabled message security in the main policy with symmetric signature and encryption policies and secure conversation tokens for both integrity and confidentiality protection.

Click **Remove bootstrap policy** to discontinue using secure conversation policy settings.

Configuring the request or response token policies:

You can configure the request and response token policies that are part of the WS-Security policy using the administrative console. Message requests token policies are applied to requests and enforced on responses to support both quality and interoperability.

Before you begin

You can configure some settings for the policies within your policy sets. The default policy sets provided in the product cannot be edited. You must create a copy of the default policy set or create a completely new policy set in order to specify the policies for it.

About this task

Use this administrative console task to define policies that specifically support security tokens and properties.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation

to learn more about the valid roles for the application server.

1. Click **Services > Policy sets > Application policy sets > *policy_set_name* > WS-Security policy**.
2. Click one of the following links:
   - **Main policy** or
   - **Bootstrap policy**
   - Click the Main policy link to specify how message security policies are applied to requests and enforced on responses to support interoperability.
   - Click the Bootstrap policy link to configure how secure conversations are established. A bootstrap policy might already be configured. If no bootstrap policy is currently configured, first ensure that you have enabled message security with symmetric signature and encryption policies and secure conversation tokens for both integrity and confidentiality protection. See Configuring the WS-Security policy.
3. Click **Request token policies** under Request Policies or **Response token policies** under Response Policies. Use this to panel to define policies that specify which types of security tokens are supported for the properties of each token type.

Results

Once you have customized the WS-Security policy with the associated properties, including the request and response token policies, you can then send and receive protect messages.

Request or Response token policies collection:

Use this page to define policies that specify supported security tokens and properties.

To view this administrative console page, complete the following:

1. Click **Services > Policy sets > Application policy sets** >*policy_set_name*.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.
4. Click **Request token policies** or **Response token policies** from the Policy Details section.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Add token type:

Specifies a list of token types that you can add to the selected token name. You can only add a token type for a custom policy set.

This option provides the following supported token types:

- Username
- X.509
- LTPA
- Custom

Delete:

Removes the selected token name. This action is only available for a custom policy set.

Token name:

Specifies the name of the token.

Type:

Specifies the token type in the Supported token types table. This list displays a token type for each token name in the list.

Version:

Specifies the version of the token in the Supported token types table. This list displays the version of each token in the list.

Token type settings:

Use the administrative console to define the details about the token types. This panel is displayed differently for each different token type. Policies can be defined that specify which types of security tokens are supported as well as properties for the token type.

To view token types for a policy set, complete the following steps:

1. Click **Services > Policy sets > Application policy sets > *policy_set_name***.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.
4. Click one of the following:
   - **Request token policies** from the Policy detail section.
   - **Response token policies** from the Policy detail section.
   - **Symmetric signature and encryption policies** from the Key symmetry section.
   - **Asymmetric signature and encryption policies** from the Key symmetry section.
5. For a Request token policy or a Response token policy, click a token from the Supported Token Types table or click the **Add Token Type** button to select the type of token to add.
6. For a symmetric signature and encryption policy or an asymmetric signature and encryption policy, click **Edit Selected Type Policy**.

This panel is displayed for each token type you are configuring or adding. It displays fields for some token types and not for others. This help panel contains all of the fields for each of the token types and describes which token is being configured for each field.

Custom token name:

For a custom token, specifies the name of the token being configured. Enter or edit the name for the custom token in this entry field.

Local name:

For a custom token, specifies the local name.

If the custom token type is used to generate a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile v1.1, use one of the values in the following table for the local name. The value you choose depends on the specification level of the Kerberos token generated by the Key Distribution Center (KDC). The table lists the values and the specification level associated with each value. For purposes of interoperability, the Basic Security Profile V1.1 standard requires the use of the local name, http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ.

Local Name Value for Kerberos Token	Associated Specification Level
http://docs.oasis-open.org/wss/oasiswss-kerberos-token-profile-1.1#Kerb erosv5_AP_REQ	Kerberos V5 AP-REQ as defined in the Kerberos specification. This value is used when the Kerberos ticket is an AP Request.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ	GSS-API Kerberos V5 mechanism token containing a KRB_AP_REQ message as defined in RFC-1964 [1964], Sec. 1.1 and its successor RFC-4121, Sec. 4.1. This value is used when the Kerberos ticket is an AP Request (ST + Authenticator).
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ1510	Kerberos V5 AP-REQ as defined in RFC1510. This value is used when the Kerberos ticket is an AP Request per RFC1510.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ1510	GSS-API Kerberos V5 mechanism token containing a KRB_AP_REQ message as defined in RFC-1964, Sec. 1.1 and its successor RFC-4121, Sec. 4.1. This value is used when the Kerberos ticket is an AP Request (ST + Authenticator) per RFC1510.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#Kerberosv5_AP_REQ4120	Kerberos V5 AP-REQ as defined in RFC4120. This value is used when the Kerberos ticket is an AP Request per RFC4120.
http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ4120	GSS-API Kerberos V5 mechanism token containing an KRB_AP_REQ message as defined in RFC-1964, Sec. 1.1 and its successor RFC-4121, Sec. 4.1. This value is used when the Kerberos ticket is an AP Request (ST + Authenticator) per RFC4120.

URI:

For a custom token, specifies the uniform resource identifier (URI).

Leave this field empty, if the custom token type is used to generate a Kerberos token as defined in the OASIS Web Services Security Specification for Kerberos Token Profile v1.1.

LTPA token name:

For an LTPA token, specifies the name of the token being configured. Enter or edit the name for the LTPA token in this entry field.

Propagate the JAAS subject:

For an LTPA token, specifies whether the associated Java Authentication and Authorization Service (JAAS) subject is propagated. Select this check box to propagate the JAAS subject. The default value is not selected. Therefore, the JAAS subject is not propagated by default.

Username token name:

For a Username token, specifies the name of the token being configured. Enter or edit the name for the user name token in this entry field.

WS-Security version:

For a Username token, specifies the version of Web services security, the WS-Security specification, that is used to secure the message transmission.

The following versions are available:

- WS-Security V1.0
- WS-Security V1.1

X.509 token name:

For a X.509 token, specifies the name of the token being configured. Enter or edit the name for the X.509 token in this entry field.

WS-Security version:

For a X.509 token, specifies the version of Web services security that is used to secure the message transmission.

The following versions are available:
- WS-Security V1.0
- WS-Security V1.1

X.509 type:

For a X.509 token, specifies the type of X.509 token being configured.

The following types are available for the X.509 token:
- X.509 Version 1. This option is available with WS-Security Version 1.1 only.
- X.509, Version 3
- X.509 PKCX7
- PKI Path Version 1

Secure conversation token: The secure conversation token is available only when using symmetric signature and encryption policies.

Key derivation requirements:

For a secure conversation token, specifies whether derived keys are required.

Require reference to secure context token issuer:

For a secure conversation token, select this option to specify a reference to the issuer of the security context token.

After selecting the **Require reference to secure context token issuer** option, specify the URI of the security context token issuer.

Require an external URI reference:

For a secure conversation token, select this option to specify that an external URI reference is required when referencing the security context token.

Transform algorithms settings:

Use this administrative console page to select the uniform resource locator (URL) for the transform algorithms that are needed to protect the message part.

To view this administrative console page, complete the following:

1. Click **Services > Policy sets > General provider policy set bindings** or **General client policy set bindings**.

2. Click the name of the binding you want to edit.
3. Click **WS-Security** policy in the Policies table
4. Click the **Authentication and Protection** link in the Main message security policy bindings section.
5. Select a signature protection in the Request message and encryption protection section or the Response message signature and encryption protection section.
6. Click the **Additional bindings > Signed part reference default** at the end of the panel.
7. Click a URL in the Transform algorithms table.

You can also get to this administrative console page by completing the following actions:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services.
3. Select **Service provider policy sets and bindings** or **Service client policy sets and bindings**.
4. Select a binding.

   Note: You must have previously attached a policy set and assigned an application specific binding before you can do this step.
5. Select **WS-Security**.

   Note: You must have previously added WS-Security to the bindings.
6. Click the **Authentication and Protection** link in the Main message security policy bindings section.
7. Select a signature protection in the Request message and encryption protection section or the Response message signature and encryption protection section.
8. Click the **Additional bindings > Signed part reference default** at the bottom of the panel.
9. Click **New** to add a transform.
10. [Optional] Click a URL in the Transform algorithms table. This step assumes that a transform algorithm has previously been configured.

URL:

Specifies the URL for the transform algorithms that are used to protect the message part.

The URLs of the supported transform algorithms are listed in the table. The recommended transform is *http://www.w3.org/2001/10/xml-exc-c14n#* and it is the Exclusive XML Canonicalization, as well as the default value provided in the URL field when a transform is added.

List of URLs for the transform algorithms
http://www.w3.org/2001/10/xml-exc-c14n#
http://www.w3.org/TR/1999/REC-xpath-19991116
http://www.w3.org/2002/06/xmldsig-filter2
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
http://www.w3.org/2002/07/decrypt#XML
http://www.w3.org/2000/09/xmldsig#enveloped-signature

Custom properties:

Specifies the custom properties name and value pair for the transform algorithms. Click **New** to add a new custom property. Click **Delete** to delete a custom property.

Name Specifies the name of a custom property that you choose to set to protect the message part.

Value Specifies the value of a custom property that you choose to set to protect the message part.

Signed part reference default bindings settings:

Use this administrative console page to configure the signed part reference general bindings and the uniform resource locator (URL) for the transform algorithms that are needed to protect the message part.

You can view and configure bindings for WS-Security using this administrative console page:

1. Click **Applications > Application Types > WebSphere enterprise applications**.
2. Select an application that contains Web services
3. Select **Service provider policy sets and bindings** or **Service client policy sets and bindings**
4. Select a binding.

   Note: You must have previously attached a policy set and assigned an application specific binding before you can do this step.
5. Select **WS-Security**.

   Note: You must have previously added WS-Security to the bindings.
6. Click the **Authentication and Protection** link in the Main message security policy bindings section.
7. Select a signature protection in the Request message and encryption protection section or the Response message signature and encryption protection section.
8. Click the **Additional bindings - Signed part reference default** at the bottom of the panel. For the custom, select an assigned message part and click **Edit**. In the custom version, there is also a reference field, **Specifies the name of the signature protection in the policy set**. Use this field to specify the name of the signature protection in the policy set.
9. Check the **Include timestamp** check box to include the timestamp when the message part is signed.
10. Check the **Include nonce** check box to include the nonce when the message part is signed.

Part reference properties:

Specifies the name of the part reference properties that include the transform algorithms used to protect the message part.

1. Check the **Include timestamp** check box to include the timestamp for the message part.
2. Check the **Include nonce** check box to include the nonce when the message part is signed.

Transform algorithms:

Specifies the uniform resource locator (URL) description for the transform algorithms. Click the URL to view the transform algorithms.

Main policy and bootstrap policy settings:

Use this page to specify how message security policies are applied to requests and enforced on responses, as defined by the main policy settings and the bootstrap policy settings. Assertions for Web Services Security (WS-Security) versions are already generated based on assertions in the policy set. If the policy set includes a Web Services Security Version 1.1 assertion, then Web Services Security Version 1.1, itself, is asserted.

To view this administrative console page, use one of the following steps:

1. Click **Services > Policy sets > Application policy sets > *policy_set_name***.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.

Message level protection:

Specifies whether message level protection, using digital signatures and encryption, is required.

Require signature confirmation

Specifies whether the signature confirmation is required. Select this check box to require a signature confirmation.

Message part protection:

Specifies whether message part protection, using digital signatures and encryption, is required.

Request message part protection
Click this link to define which request message parts you want protected and how that protection is provided.

Response message part protection
Click this link to define policies that specify which response message parts you want protected and how that protection is provided.

When the Message level protection check box is cleared, the link to Request message part protection is disabled, because the configuration information associated with message level security is removed when message level protection is cleared.

Key symmetry – Use symmetric tokens:

Specifies whether to use symmetric tokens. Select this radio button to use symmetric tokens. You can then configure symmetric tokens using the **Symmetric signature and encryption policies** link. Click this link to access the Symmetric signature and encryption policies panel where you can create the trust context in which to use symmetric tokens. Using the same token for signing and validating messages and encrypting and decrypting messages provides higher performance than can be achieved with asymmetric tokens. Use symmetric tokens within a trust context. If a custom Kerberos token type is used, you must select the Use symmetric tokens option.

Key symmetry – Use asymmetric tokens:

Specifies whether to use asymmetric tokens. Select this button to use asymmetric tokens. You can then configure asymmetric tokens using the **Asymmetric signature and encryption policies** link. Click this link to access the Asymmetric signature and encryption policies panel where you can create the trust context (message integrity and confidentiality) in which to use asymmetric tokens. Specify which token type to use for the initiator and recipient signature as well as the initiator and recipient encryption.

Include time stamp in security header:

Specifies whether to use a time stamp in the header. Select this check box to include a time stamp in the header. You can then specify where in the header to place the time stamp by using the **Security header layout** radio buttons.

Security header layout:

Specifies the layout rules for the security header.

You can use the following radio buttons for the security header layout:

Strict: declarations must precede use
The declarations in the header must precede the use.

Layout (Lax): order of contents can vary
The order of contents in the header can vary.

Lax but timestamp required first in header
The timestamp must be first in the header, but the order of the remaining elements can vary.

Lax but timestamp required last in header
The timestamp must be last in the header, but the order of the remaining elements can vary.

Policy details:

Specifies links for accessing the request token policies, response token policies, and algorithms for asymmetric tokens. Click these links to view token policies and cannonicalization algorithms that are supported. Algorithms are used to reconcile XML differences.

Request token policies:

Click this link to define policies that specify which types of supporting authentication tokens are used in the request and the properties of those token types.

Response token policies:

Click this link to define policies that specify which types of supporting authentication tokens are used in the response and the properties of those token types.

Algorithms for symmetric or asymmetric tokens:

Links to a view of available algorithms. Click this link to view the cryptographic and cannonicalization algorithms that are supported. Algorithms are used to reconcile XML differences.

Asymmetric signature and encryption policies settings:

Use this page to create the trust context, message integrity and confidentiality, to use asymmetric tokens. You can create the trust context by specifying which token type to use for the initiator and recipient signature as well as the initiator and recipient encryption.

To view this administrative console page complete the following actions:
1. Click **Services > Policy sets > Application policy sets > *policy_set_name* > WS-Security policy type**.
2. Click **Main policy** or **Bootstrap policy**.
3. Click the **Asymmetric signature and encryption policies** link.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Message integrity policies – Initiator signature token:

Specifies the token type of the initiator signature token. To add a token type or change the current token type that is displayed in the **Initiator signature token** field, or to edit the displayed token type, click **Action**.

Message integrity policies – Recipient signature token:

Specifies the token type of the recipient signature token. To add a token type or change the current token type that is displayed in the **Recipient signature token** field, or to edit the displayed token type, click **Action**.

Message confidentiality policies – Use the same token types that are used for integrity protection:

Specifies whether the token type set for initiator signature token and recipient signature token are used for the initiator encryption token and the recipient encryption token. When the box is checked the fields are empty and are cleared of data when **Ok** or **Apply** is selected.

Message confidentiality policies – Initiator encryption token:

Specifies the initiator encryption token type. To add a token type or change the current token type that is displayed in the **Initiator encryption token** field, or to edit the displayed token type, click **Action**.

Message confidentiality policies – Recipient encryption token:

Specifies recipient encryption token type. To add a token type or change the current token type that is displayed in the **Recipient encryption token** field, or to edit the displayed token type, click **Action**.

Action:

Specifies an option for each of the signature and encryption token fields. Use the **Action** button to change, delete, add, or edit the listed token type.

The Action button lists supported token types and provides the following options:

Edit selected type policy
Opens a page to edit the token type shown in the signature or encryption token fields.

Delete selected type policy
Removes the token type from the signature or encryption token fields.

Change to custom type
Opens the Custom type page to specify the uniform resource identifier (URI) for a custom token type.

Add custom type
Adds the custom type entry in the signature or encryption token fields.

Change to X.509
Changes the listed token type to X.509.

Add X.509
Adds the X.509 token type.

When you change the token type, any values you specified for the former token type are lost and the default values for the newly assigned token type are used.

Symmetric signature and encryption policies settings:

Use this page to create the trust context to use symmetric tokens. Using the same token for signing and validating messages and encrypting and decrypting messages increases performance. Use symmetric tokens within a trust context.

To view this administrative console page, complete the following options:

1. Click **Services > Policy sets > Application policy sets**.
2. Select a ***policy_set_name*** in the policy sets table that contains WS-Security content.
3. Click **WS-Security** in the policies table.
4. Click the **Main policy** link or the **Bootstrap policy** link.
5. Click the **Symmetric signature and encryption policies** link.

Message Integrity – Token type for signing and validating messages:

Specifies the current token type used for signing and validating messages.

To change the current token type that is displayed in the **Token type for signing and validating messages** field or to edit the displayed token type, click **Action**.

Message Confidentiality – Use same token type for confidentiality that is used for integrity:

Specifies whether the token type set for signing and validating messages is also used for encrypting and decrypting messages. For a Kerberos token, message confidentiality uses the same token that is used for the message integrity.

If you select this check box, then the **Token type for encrypting and decrypting messages** field is blank. If you clear this check box, then a different token can be used for message confidentiality.

Message Confidentiality – Token type for encrypting and decrypting messages:

Specifies the current token type that is used for encrypting and decrypting messages.

To change the current token type that is displayed in the **Token type for encrypting and decrypting messages** field or to edit the displayed token type, verify that the **Use the same token type for confidentiality that is used for integrity protection** check box is cleared, and click **Action**.

Action:

Enables the token type selected to be changed or edited.

The **Action** button lists supported token types and provides the following options:

Edit selected type policy
Opens a page to edit the token type for signing or encrypting fields.

Change to Secure Conversation
Changes the token type to Secure Conversation.

Change to custom type
Opens the custom type page to specify the Uniform Resource Identifier (URI) for a custom token type.

When you change the token type, any values that you specified for the former token type are lost, and the default values for the newly assigned token type are used.

Algorithms settings:

Use this page to view the supported cryptographic and cannonicalization algorithms. Algorithms are used to reconcile XML differences.

To view this administrative console page:

1. Click **Services > Policy sets > Application policy sets > *policy_set_name***.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.
4. Click the **Algorithms for symmetric tokens** link or the **Algorithms for asymmetric tokens** link.

This administrative console panel applies only to Java API for XML Web Services (JAX-WS) applications.

Algorithm suite:

Specifies the supported algorithms that are required for performing cryptographic operations with symmetric or asymmetric key-based security tokens.

All of the algorithm values in this field specify an algorithm suite. Algorithm suites and the values they each represent are detailed in the *Web Services Security Policy Language (WS-SecurityPolicy) July 2005 Version 1.1* specification. Select a supported algorithm from the following list:

- Basic256
- Basic192
- Basic128
- TripleDes
- Basic256Rsa15
- Basic192Rsa15
- Basic128Rsa15
- TripleDesRsa15
- Basic256Sha256
- Basic192Sha256
- Basic128Sha256
- TripleDesSha256
- Basic256Sha256Rsa15
- Basic192Sha256Rsa15
- Basic128Sha256Rsa15
- TripleDesSha256Rsa15

This table defines values for the components for each algorithm suite.

Table 8. Algorithm suite components

Algorithm Suite	Digest	Encryption	Symmetric Key Wrap	Asymmetric Key Wrap	Encryption key Derivation	Signature key Derivation	Minimum Symmetric Key Length
Basic256	Sha1	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192	Sha1	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128	Sha1	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128
TripleDes	Sha1	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Rsa15	Sha1	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Rsa15	Sha1	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Rsa15	Sha1	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesRsa15	Sha1	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192
Basic256Sha256	Sha256	Aes256	KwAes256	KwRsaOaep	PSha1L256	PSha1L192	256
Basic192Sha256	Sha256	Aes192	KwAes192	KwRsaOaep	PSha1L192	PSha1L192	192
Basic128Sha256	Sha256	Aes128	KwAes128	KwRsaOaep	PSha1L128	PSha1L128	128
TripleDesSha256	Sha256	TripleDes	KwTripleDes	KwRsaOaep	PSha1L192	PSha1L192	192
Basic256Sha256Rsa15	Sha256	Aes256	KwAes256	KwRsa15	PSha1L256	PSha1L192	256
Basic192Sha256Rsa15	Sha256	Aes192	KwAes192	KwRsa15	PSha1L192	PSha1L192	192
Basic128Sha256Rsa15	Sha256	Aes128	KwAes128	KwRsa15	PSha1L128	PSha1L128	128
TripleDesSha256Rsa15	Sha256	TripleDes	KwTripleDes	KwRsa15	PSha1L192	PSha1L192	192

When using a Kerberos custom token based on the OASIS Web Services Security Specification for Kerberos Token Profile V1.1, only Aes128, Aes256, and TripleDes encryption-based algorithm suites are supported.

Cannonicalization algorithm:

Specifies whether to use inclusive or exclusive cannonicalization.

The following supported cannonicalization algorithms are available in this list:

- Exclusive cannonicalization
- Inclusive cannonicalization

The default value is `Exclusive cannonicalization`.

XPath version:

Specifies the version of the XPath filter to use.

The following supported XPath versions are available:

- XPath 1.0
- XPathfilter 2.0

The `XPathfilter 2.0` version is the default value.

Use security token reference transformation:

Specifies whether the security token reference is transformed. Indicate whether the security token reference transform is either `True` or `False`.

Message part protection settings:

Use this page to define the message parts that you want protected and how that protection is provided.

To view this administrative console page, complete the following actions:

1. Click **Services > Policy sets > Application policy sets** >*policy_set_name*.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.
4. Click the **Request message part protection** link or the **Response message part protection** link from the Message Part Protection section.

Integrity protection – Signed parts:

Specifies the signed parts that you have added for message integrity protection. You can select parts to edit or delete, or add new signed parts.

Button	Resulting Action
Add	Specifies each element of a new signed part and to add those elements to the Signed parts listing.
Edit	Loads the specific part you selected for editing.
Delete	Removes the selected signed part from the list.

Confidentiality protection – Encrypted parts:

Specifies the encrypted parts that have been added for message confidentiality protection. You can select parts to edit or delete, or add new encrypted parts.

Button	Resulting Action
Add	Enables you to specify each of the elements of a new encrypted part and to add those elements to the Encrypted parts listing.
Edit	Loads the specific part you selected for editing.
Delete	Removes the selected encrypted part from the list.

Signed part settings:

Use this page to define the elements of a signed part. Signed parts are used to protect message integrity and, in this case, the signed parts are being defined as part of the policy set process.

To view this administrative console page, complete the following actions:

1. Click **Services > Policy sets > Application policy sets > *policy_set_name***.
2. Click the **WS-Security** policy in the Policies table.
3. Click either the **Main policy** link or the **Bootstrap policy** link.
4. Click the **Request message part protection** link or the **Response message part protection** link in the Message Part Protection section.
5. In the Integrity protection section, complete one of the following actions:
   - Click **Add** to add a new signed part.
   - Select an existing signed part and click **Edit**.

Name of part to be signed:

Specifies the name of this set of one of more message parts that you have selected to sign. The name you choose is a label and must be unique within the Response message part protection or Request message part protection collections for this WS-Security policy.

Elements in part:

Specifies a list of the message elements included in the signed part. The **Elements in part** field contains a listing of message elements that are included in this signed part to provide message integrity.

Click **Add** to add an element to the signed part of the message. To remove a message element from a signed part of a message, first click the selection box next to the element to be removed, then click **Remove**. Use the **OK**, **Apply**, **Reset** or **Cancel** buttons for the text entry fields. The QName or the Xpath expression value is required and can be edited at any time, such as when adding a new element, or after the element is added.

Protect message body:

Specifies if the message body is protected in this part. To protect the message body in this part, click **Protect message body**.

XPath expression:

Specifies if the displayed XPath expression is used as the method for specifying that a specific element is included in this part.

Select **XPath** from the **Add** menu list and provide an expression in the new XPath entry that is displayed in the table. Any Xpath expression row on the table that has no corresponding value is removed when you click **OK** or **Apply**.

QName for SOAP header elements only:

Specifies the Qname type for a namespace value for the SOAP header element that you want to encrypt. To encrypt a SOAP header element, select **Qname** and provide the **namespace** and optionally the **localname** of the SOAP header element in the **Value** field. When specifying the Qname, if using the optional localname, a comma must be inserted between the namespace and the localname, for example **<namespace>,<localname>**. If the **localname** is omitted, all SOAP header elements with the specified **namespace** are encrypted. To use the **Qname** selection method, the SOAP header elements must be the immediate children of the SOAP header. Any Qname row in the table that has no corresponding value is removed when you click **OK** or **Apply**.

Note:
You cannot select header elements that are sub-elements of other elements in the SOAP header using **Qname**. In this case, you must use an **Xpath expression** to select these header elements.

Encrypted message part settings:

Use this page to define the elements of an encrypted part of a message. Encrypted parts are used to protect message confidentiality, and in this case, the encrypted parts are being defined as part of the policy set process. A message part is a named set of one or more message elements.

To view this administrative console page, complete the following actions:

1. Click **Services > Policy sets > Application policy sets > *policy_set_name***.
2. Click the **WS-Security** policy in the Policies table.
3. Click the **Main policy** link or the **Bootstrap policy** link.
4. Click the **Request message part protection** link or the **Response message part protection** link in the Message Part Protection section.
5. In the Confidentiality protection section, you can perform any of the following:
   - Click **Add** to add a new encrypted part.
   - Select an existing encrypted part, and click **Edit**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Name of part to encrypt:

Specifies the name of the set of one or more message parts that you have selected to encrypt. The name you choose is a label and must be unique within the Response message part protection or Request message part protection collections for this WS-Security policy.

Elements in part:

Specifies a list of the message elements that are included in the encrypted part. The **Elements in part** field contains a listing of message elements that are included in this encrypted part to provide message confidentiality.

Click **Add** to add an element to the encrypted part of the message. To remove a message element from an encrypted part of a message, first click the selection box next to the element to be removed, then click **Remove**. The value of the Qname namespace, or the Xpath expression, is required and can be edited at any time, while adding a new element or after the element is added.

Body Specifies the body of the message part.

Qname for SOAP header elements only
Specifies the Qname type for a namespace value for the SOAP header element that you want to encrypt. To encrypt a SOAP header element, select **Qname** and provide the **namespace** and

optionally the **localname** of the SOAP header element in the **Value** field. When specifying the Qname, if using the optional localname, a comma must be inserted between the namespace and the localname, for example **<namespace>,<localname>**. If the **localname** is omitted, all SOAP header elements with the specified **namespace** are encrypted. To use the **Qname** selection method, the SOAP header elements must be the immediate children of the SOAP header. Any Qname row in the table that has no corresponding value is removed when you click **OK** or **Apply**.

Note: You cannot select header elements that are sub-elements of other elements in the SOAP header using Qname. In this case, you must use an Xpath expression to select these header elements.

Xpath expression
Specifies if the displayed Xpath expression is used as the method for specifying that a specific element is included in this part. Select **XPath** from the **Add** menu list, and provide an expression in the new XPath entry that is displayed in the table. Any Xpath expression row on the table that has no corresponding value is removed when you click **OK** or **Apply**.

Enabling policies for policy sets using the administrative console

Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

Before you begin

To enable a policy for a policy set, be sure the policy is listed in the policy set and shown in the **Disabled** state in the **State** column of the **Policies** table on the Policy set settings page.

About this task

To enable a policy in a policy set, use the administrative console.

1. Click **Services > Policy sets > Application policy sets** > ***policy_set_name*** or **Services > Policy sets > System policy sets** > ***policy_set_name***. The Policy Set Settings page displays a listing of available policies in the **Policies** table for the policy set selected. If this table contains no policies to enable, no policies exist for the policy set of interest. In this case, you must add the policies to the policy set.
2. Click the **Select** box beside the disabled policy that you want to enable. You can select multiple policies if you want to enable more than one.
3. Click the **Enable** button. The **State** column of the **Policies** table is updated to display the selected policy as enabled.

Results

You have enabled a policy for the selected policy set.

What to do next

If the policy is not listed in the Policies table, it cannot be enabled and must be added. You can modify the policy after it is enabled.

Related concepts

“Web services policies” on page 952
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

Related tasks

“Adding policies to policy sets using the administrative console” on page 903
You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

“Deleting policies from policy sets using the administrative console” on page 904
You can use the administrative console to delete policies from policy sets. Deleting the policy removes the policy that further defined the rules governing the policy set.

“Disabling policies from policy sets using the administrative console”
You can have policies listed in a policy set that are in the enabled state so that they are currently included in the policy set. You can disable a policy from being included in a policy set without deleting it from the policy set. You might want to do this if you want the policy included in the policy set in the future. You can use the administrative console to change this setting.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Disabling policies from policy sets using the administrative console

You can have policies listed in a policy set that are in the enabled state so that they are currently included in the policy set. You can disable a policy from being included in a policy set without deleting it from the policy set. You might want to do this if you want the policy included in the policy set in the future. You can use the administrative console to change this setting.

Before you begin

To disable a policy for a policy set, be sure the policy is listed in the policy set and shown in the **Enabled** state in the **State** column of the **Policies** table on the Policy set settings page. If the policy is not listed, it might have been deleted and you must add it again and then disable it.

About this task

You can disable a policy in a policy set, from the menu of the administrative console.

1. Click **Services > Policy sets > Application policy sets** >***policy_set_name*** or **Services > Policy sets > System policy sets** >***policy_set_name***. Clicking a policy set name from the listing in the policy sets collection opens the Policy set settings panel for that policy set. This panel displays a listing of available and enabled policies in the **Policies** table for the policy set you selected. If this table contains no policies, there are no existing policies to disable.
2. Click the**Select** box beside the enabled policy to be disabled. You can select multiple policies if you want to disable more than one.

3. Click **Disable**. The **State** column of the **Policies** table is updated to display the selected policy as disabled. The policy was not deleted from the policy set and is still available to be enabled.

Results

You have disabled a policy for the selected policy set.

What to do next

You could now enable other policies or add or modify existing policies for this policy set.

Related tasks

“Adding policies to policy sets using the administrative console” on page 903
You can use the administrative console to add policies to policy sets. Adding and configuring policies to a policy set further defines the rules governing the policy set.

“Deleting policies from policy sets using the administrative console” on page 904
You can use the administrative console to delete policies from policy sets. Deleting the policy removes the policy that further defined the rules governing the policy set.

“Enabling policies for policy sets using the administrative console” on page 950
Policies can be listed in a policy set in the disabled state so that they are not currently included in the policy set. You can enable a policy to be included in a policy set using the administrative console.

“Web services policies”
Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

“Managing policy set attachments using the wsadmin tool” on page 518
Use the wsadmin tool to manage your policy set attachment configurations. You can use the Jython or Jacl scripting language to list all attachments and attachment properties, add or remove resources for an existing attachment, and transfer attachments across policy sets.

Related reference

“Application policy sets collection” on page 954
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

“Application policy set settings” on page 955
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

Web services policies

Policies define the type of Web service policy based on the quality of service type. Policies are initially set with default settings but the attributes can be edited and changed.

Provided policies include:

WS-Addressing
: Based on the Worldwide Web Consortium (W3C) WS-Addressing specifications for Web services. This family of specifications provide transport-neutral mechanisms to address Web services and to facilitate end-to-end addressing. This specification provides asynchronous support.

WS-Security
: Based on the WS-Secure Conversation (WS-SC) and WS-Security specifications along with the associated token profiles. The WS-Security specification and its associated token profiles define a way to send security tokens and provide message integrity and confidentiality. The WS-Secure Conversation specification establishes a secure context, based on shared keys, for the client and server to use for a series of messages. This standard provides a framework across organizations

that defines how to secure the entire conversation. Use the WS-Security policy to define how the SOAP messages are secured. It has options such as:

- which message parts are signed and encrypted
- the tokens types to be included
- whether to use symmetric or asymmetric cryptography

You can also use WS-Security policies to define the bootstrap policy that is used to acquire security context tokens. Security context tokens are used by secure conversation.

WS-Reliable Messaging (WS-RM)
This specification enables the sender and receiver to assure the quality of services in a set of messages. It helps the application developer deal with latency issues, maintenance interruption, and other problems that prevent messages from being completed. This quality assurance is critical for stateful applications.

WS-Transaction
This specification provides support for coordination of atomic transactions or business activities for Web services applications. You can enable WS-Transaction on both the client (outbound) and server (inbound) side by attaching a policy set that enables the WS-Transaction policy as part of the policy set. This policy is based on the WS-AtomicTransaction specification and the WS-BusinessActivity specification, together with the WS-Coordination specification.

HTTP Transport
The HTTP transport policy applies the HTTP features and HTTP connections polices to outbound messages. The response listener policy is enforced on inbound messages.

SSL Transport
Provides SSL transport security for the HTTP protocol with Web services applications.

Related information

WS-Policy working group

OASIS WS-SX Technical Committee

Exporting policy sets using the administrative console

You can export policy sets between a client and a provider or between servers using the administrative console.

Before you begin

Before you begin this task, select the policy set to be exported. Read about the application policy sets collection and exporting policy sets to client or server environments using the wsadmin tool.

About this task

Static export is used in development environment to exchange a policy sets between a client and a provider. You can also export between servers. The exported format is a .zip file.

1. From the administrative console navigation, click **Services > Policy sets > Application policy sets** or **Services > Policy sets > System policy sets**. The Application policy sets collection page displays a listing of the custom and default policy sets. Custom policy sets are displayed only if you have created them. If you have not created a custom policy set, then only the default policy sets are displayed.
2. From the Application policy sets panel, select a policy set and click **Export**.
3. Locate the export archive file. From the Export policy set archive file settings panel, click the archive file to export.
4. Choose a location to export the file. Do not export the file to a browser.

Results

When you finish this task, the policy set archive file has been exported to the location you specified.

Example

If you have a policy set, `ABC_ps` and you want to move it from `ServerA` to `ClientA`, you can use the export function. The export policy set can be used by importing it into another application server. Read about the command task for importing an exported policy set zip file.

What to do next

You can import the policy set to reuse it.

Related tasks

“Importing and exporting policy sets to client or server environments using scripting” on page 539
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to export and import application or system policy sets for Web services. The exportPolicySet command creates an archive file based on the policy set configuration, and the importPolicySet command imports a default policy set or policy set from an archive file.

Related reference

“Application policy sets collection”
Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

Application policy sets collection

Use this page to manage policy sets. You can create, copy, export, and import policy sets. You can also view or delete existing policy sets. You can use policy sets, or assertions that define services, to simplify your Web services configuration because policy sets group security and other Web services settings into reusable units.

To view this administrative console page, click **Services > Policy sets > Application policy sets**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Name

Specifies a list of available policy sets. Click a policy set name to access the details panel for that policy set. Use the buttons on the page to create a new policy set or manage the existing policy sets. You can also import a policy set to display in the list of available policy sets.

The following list of buttons are available to you for creating and managing policy sets:

Button	Resulting action
New	Creates a new policy set. This option opens an empty policy set and allows you to provide additional details, such as the name, description, and policy configuration options.
Delete	Removes the selected policy set.
Copy	Creates a copy of the policy set that is selected from a list. After you create a copy, you can provide a name and a description for the copy. You must also specify whether to transfer attachments and binding information, where these options are available, from the original to the copy.

Button	Resulting action
Import	Imports a policy set. This is a menu item with the option of importing a policy set from a default repository or a selected location. You can select and import the default policy sets from the default repository. The default repository for the import function in the administrative console is the directory that contains the default policy sets. Initially, up to four default, pre-configured policy sets are exposed. However, there are other default policy sets that are available to import when you select **Import > From Default Repository**. The administrative console also displays the default policy sets in a list that includes descriptions, that you can use to select the policy set that you want to import.
Export	Exports the selected policy set to an archive file.

Editable
This column shows whether the policy set is a user-defined, custom policy set that can be edited or whether the policy set is a default policy set that is not editable.

Values displayed in this field are `Editable` or `Not editable`. Even though a policy set is identified as not editable, it is deletable. For example, you cannot edit information for the default system policy set, but you can delete the policy set. You can change the properties for a default, not editable policy set by copying it, and then modifying the properties of the copy. For more information on modifying a default policy set, see copy of default policy set and bindings settings document.

Data type: String
Default: Not editable

Description
Provides a brief description of the application policy sets that currently exist. You cannot edit information for the default application policy sets; however, you can delete the policy sets. The list of contained policies for each policy set that it relates to are displayed in the Name column. The contained policies that are displayed are only those that are enabled. From this panel, the field is not editable.

For custom policy sets that you create or modify, you can edit this description when you create or customize policy sets on the details panel. The description is the same one that you entered in the **Description** field when you created that policy set. For default policy sets, the included and enabled policies appear at the top followed by a bulleted list of the key attributes of the policy set.

Application policy set settings
Use this page to view, create, enable or disable your policy sets. You can use policies, or assertions that define services, to simplify your Web services configuration.

To view this administrative console page, click one of the following paths:
- **Services > Policy sets > Application policy sets** >*policy_set_name*
- **Services > Policy sets > Application policy sets** > **New**.

Depending on your assigned security role when security is enabled, you might not have access to text entry fields or buttons to create or edit configuration data. Review the administrative roles documentation to learn more about the valid roles for the application server.

Note: You can only edit the fields on this page if you open this page for a custom policy set. You cannot edit default policies.

Policy set name
Specifies the name of a selected policy set. This field is empty if you are creating a new policy set and you can enter a policy set name. If you select an existing policy set from the Policy set page, then this field displays the name of that policy set. This field is also not editable for existing custom policy sets and default policy sets.

Description
Specifies a brief description of the policy set being viewed, modified or created. This field displays a brief description of the policy set displayed in the Policy set name field or is empty if a new policy set is being created. You can edit this field for custom policy sets only.

Policies
The Policies table displays a list of policies that are contained in the policy set. This table initially contains no policies when you are creating a new custom policy set. If this is a default policy set, then you cannot add, delete, enable or disable the policy. The configuration of a policy in a default policy set cannot be altered but you can view a policy by clicking on the policy name link. If the policy is a custom policy, you can modify the configuration of the policy by clicking the policy name link. You can alter the policies contained in a custom policy set using the following buttons:

Button	Resulting Action
Add	Provides a list of valid policies, instances of which are not already included in the collection, that can be added to the policy set collection. The following policies are available: • WS-Addressing • WS-Security • WS-ReliableMessaging • WS-Transaction • HTTP transport • JMS transport • SSL transport
Delete	Removes the selected policy from the policy set and from the listing in the Description column of the table.
Enable	The policy is enabled for the policy set and displayed as enabled in the State column.
Disable	The policy is disabled for the policy set. The policy remains in the list but the State field shows the policy as Disabled.

Policies - Policy
Specifies the name of a policy. This field is empty if you are creating a new policy set. If you select an existing policy, then this field displays the name of that policy. If this policy is a default policy, then this field is read only. If you are creating a new policy, then this field is empty and you can enter a policy name.

Policies - State
Specifies an Enabled or Disabled status for each policy in the policy set. You cannot change the state for default policy sets. To change the state of the policy, select the policy and click **Enable** or **Disable**. The state is refreshed in this column to reflect your change.

If you disable a policy, you are temporarily removing the policy from the policy set without losing the configuration details for that policy. When a policy is disabled, you can easily enable the policy again by selecting **Enable**. If the policy is deleted, the configuration for that policy no longer exists. After you delete a policy, to enable that policy in a policy set, you must add the policy to the policy set and enter the configuration details for that policy.

Policies - Description
Specifies a brief description of each policy in the policies table. The product does not support custom policies; therefore, this field is not editable. You cannot create a new policy.

Attachments – Attached applications
Click this link to search for applications to which this policy set is attached. You can also find applications that contain service resources to which the policy set is attached.

Search attached applications collection
Use this page to search for applications and other resources that are attached to a specific policy set or to search for applications and other resources that have attached service resources.

To view this administrative console page, complete the following actions:

1. Click **Services > Policy sets > Application policy sets** >*policy_set_name*.
2. Click **Attached applications** link in the Additional Properties section.

Name
Specifies a list of applications that match the search text. The application names that are displayed in the Name column are either attached explicitly to the specified policy set or have service resources attached to this policy set.

To alter the policy set that is attached to an application, select an application, and click a button to enable the following options:

Button	Resulting Action
Detach Policy Set	Detaches the current policy set from the selected application or applications. This action also detaches any attached application service resources. This action does not detach other policy sets from the application or application service resources.
Replace Policy Set	Displays a list of policy sets that can be attached to the selected application or applications and any contained attached service resources. The policy set is replaced with the one selected. This action does not replace other policy sets that are attached to resources in the application that you select.

Click the application name to complete actions such as attaching policy sets to the application, its services, or its endpoints.

Securing message parts using the administrative console
If you are working with policy sets, then you can secure message parts using the administrative console. To secure message parts with WS-Security using policy sets, you must define the elements for the message parts to be protected in the WS-Security policy within a policy set.

Before you begin

Before you can start this task, you must have a policy set defined for your application or service artifact. Also, if none of the default policy sets contain the necessary policy definitions, then you must create a custom policy set with the necessary definitions.

About this task

This task assumes that you are using policy sets and you want to secure message parts within that context.

1. Open the administrative console.
2. Select the policy set containing the message parts that you want to secure.
   - To secure message parts using application policy sets click **Services > Policy sets > Application policy sets**.
   - To secure message parts using system policy sets click**Services > Policy sets > System policy sets**.
3. Select the policy set that you want to use.
4. If the WS-Security policy is not listed, then click **Add** and select that policy from the list.
5. Click the **WS-Security** link.
6. Click **Main policy** or **Bootstrap policy**. The bootstrap policy is available when Secure Conversation is used. If you want to use the bootstrap policy, then select the SecureConversation policy set in step three.
7. Make sure that Message level protection is selected, then click **Request message part protection** or **Response message part protection**. When the Message level protection checkbox is unchecked, the link to Response message part protection is not available, because the configuration information associated with message level security is removed when Message level protection is deselected.
8. Click **Add** for either Encrypted parts or Signed parts depending on the level of security that you want.
9. Specify a part name and add the elements to be signed or encrypted, or both. The elements can be the message body, XPath expression, or a QName which is for SOAP header elements only. Click **OK**. Recommendation for when to use QName or XPath: If you are encrypting or signing SOAP headers, you can use QName to select which SOAP headers to be signed or encrypted.

   Note: The elements must be a direct child of the SOAP headers.

   If you wanted to sign and encrypt other elements in the SOAP message, then you can use XPath expression. Use this XPath example to select, `MyElement` in a namespace, `http://xyz.acme.com` with `MyHeader`, `http://acme.com`.

```
/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='Envelope']/*[namespace-uri()=
'http://www.w3.org/2003/05/soap-envelope' and local-name()='Header']/*[namespace-uri()='http://acme.com' and local-name()=
'MyHeader']/*[namespace-uri()='http://xyz.acme.com' and local-name()='MyElement']
```

10. Repeat steps 8 and 9 to sign or encrypt each message part.
11. To save your changes to the master configuration, click **Save**.

Results

When you finish this task, you have configured the policy set that contains the quality of service definitions required for signing and encrypting message parts.

Example

If you have the policy set, **myPolicy** and you want to specify request message bodies that must be signed, you can perform the following:

1. Locate the policy set in the **Services > Policy sets > Application policy sets** collection and click the policy set name.
2. Click the **WS-Security** link. If the link does not exist, click **Add** and then select **WS-Security** from the list.
3. Click **Main policy** > **Request message part protection**
4. Click **Add** under the Integrity protection and Signed parts section.
5. Specify the name, `messageBody`.
6. Select **Protect message body**, click **Add Specified Elements**, and click **OK**.
7. Click **Save** to save your changes to the master configuration.

What to do next

You can proceed to signing and encrypting message parts using policy sets.

Related concepts

"Web services policy set bindings" on page 829
A set of bindings is a named object that is associated with a specific policy set and service resource that is attached to the policy set.

Encrypted SOAP headers
The encrypted header element provides a standard way of encrypting SOAP headers. As one of the extensions to the OASIS SOAP message security specification, the encrypted header element indicates that the responder has processed the request. Encrypting SOAP headers and parts help to provide more secure message-level security.

Related tasks

"Signing and encrypting message parts using policy sets" on page 860
With Web services, you can sign message parts, encrypt message parts, or both, based on the quality of service defined for a policy set. You can accomplish these actions by defining the binding information in a custom attachment binding.

"Creating application specific bindings for policy set attachment" on page 838
After you attach a policy set to a service artifact such as an application, service, or endpoint, you can define application specific bindings for the attached policy set.

"Modifying default bindings at the server level for policy sets" on page 855
You can define default bindings for HTTP transport, JMS transport, Secure Sockets Layer (SSL) transport, WS-Addressing, WS-ReliableMessaging, and WS-Security policies.

"Reassigning bindings to policy sets" on page 856
After you create a custom attachment binding, you can reassign that binding to another service artifact if necessary. You can reset a service artifact, such as an application, service, or endpoint to use the inherited bindings or default bindings.

"Configuring the WS-Security policy" on page 933
When working with policy sets in the administrative console, you can customize policies to ensure message security. The WS-Security policy can be configured to apply a message security (WS-Security) profile to requests. Message security policies are applied to requests and enforced on responses to support interoperability.

Related reference

"Service client policy set and bindings collection" on page 763
Use this page to attach and detach policy sets to an application, a service client, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use existing bindings for an attached policy set. You can view or change whether the client uses the policy of the service provider.

"Service provider policy sets and bindings collection" on page 746
Use this page to attach and detach policy sets to an application, a service provider, its endpoints, or operations. You can select the default bindings, create new application-specific bindings, or use bindings that you created for an attached policy set. You can view or change whether the service provider can share its current policy configuration.

"Policy set bindings settings" on page 866
Use this page to view or define general or application specific bindings configuration information that is specific to a system for policies that you can associate with the selected policy set. Use the links on this page to work with bindings for each specific policy.

"Policy set bindings settings for WS-Security" on page 868
Use this page to view, define or configure general bindings and application specific properties for the WS-Security policy. You can configure the main policy or the secure conversation bootstrap policy by editing the general bindings.

"WS-Security authentication and protection" on page 878
Use the links on this page to configure authentication, signature, and encryption information that the policy requires.

"Caller settings" on page 895

Use this page to configure the caller settings. The caller specifies the token or message part that is used for authentication.

"Message expiration settings" on page 899
Use this page to define settings for message expiration, if and when messages expire. When you specify message expiration, the message expires after the specified interval of time passes.

"Actor roles settings" on page 899
Use this page to define settings for SOAP actor roles. The SOAP actor, also known as the SOAP role, defines the intermediary or ultimate recipient of a message.

"Keys and certificates" on page 869
Use this page to link to key and certificate binding configuration panels. This panel defines key and certificate bindings for JAX-WS Web services only. These keys and certificates can be centrally managed by the product or in an external keystore.

Related information

"Web Services Addressing policy set binding" on page 900
Use this page to modify the endpoint reference binding for Web Services Addressing (WS-Addressing). The product enforces this binding on JAX-WS Web service applications that use WS-Addressing. This panel applies only to the Network Deployment version of the product.

Web services policy sets

Policy sets are assertions about how services are defined. They are used to simplify your quality of service configuration for Web services.

Note: You can use policy sets only with Java API for XML-Based Web Services (JAX-WS) applications. You cannot use policy sets with Java API for XML-based RPC (JAX-RPC) applications.

Policy sets combine configuration settings, including those for transport and message level configuration, such as WS-Addressing, WS-ReliableMessaging, and WS-Security.

There are two main types of policy sets; application policy sets and system policy sets. Application policy sets are used for business-related assertions. These assertions are related to the business operations that are defined in the Web Services Description Language (WSDL) file. System policy sets, on the other hand, are used for non-business-related system messages. These messages are not related to the business operations that are defined in the WSDL, but instead refer to messages that are defined in other specifications which apply qualities of service (QoS). Such QoS are the request security token (RST) messages that are defined in WS-Trust, or create sequence messages that are defined in WS-Reliable Messaging metadata exchange messages of the WS-MetadataExchange.

Policies are defined based on a quality of service. Policy definition is typically based on WS-Policy standard language, for example, the WS-Security policy is based on the current WS-SecurityPolicy from the Organization for the Advancement of Structured Information Standards (OASIS) standards.

An instance of a policy set consists of a collection of policies. For example, the WS-I RSP default policy set consists of instances of the WS-Security, WS-Addressing, and WS-ReliableMessaging policy types. A policy set is identified by a unique name that is unique across the cell. An empty policy set is a policy set with no policies defined.

You can use a default policy set after it is imported. If you want to change the properties for a default, not editable policy set, you need to copy the policy set to create an editable version to modify. See copy of default policy set and bindings settings. You can perform the following actions on policy sets:

- create
- copy
- edit
- delete

- attach to service resources like applications
- detach from service resources like applications
- export
- import

Note which functions you can configure using policy sets and the relationship of the security information that is configured. A set of default policy sets are included that you can import; then copy and rename for reuse. You can use a default policy set after it is imported, but if you want to change any of the settings, you need to copy the policy set to create an editable version. The configuration can then be altered and customized on the copy.

Note: You can only copy and customize policy sets using the administrative console or administrative commands. Policy sets do not function correctly if they are copied manually.

On the application server, policy sets are stored at the cell level. Policy sets are centrally located so that they are available to all applications on the server.

The following application policy sets are installed on the base or network deployment (ND) profile by default: WS-I RSP or WS-I RSP (ND), Username WSSecurity default, and WSHTTPS default. The WS-I RSP (ND) is installed in a network deployment environment.

The following policy sets are ready for you to use as is.
- LTPA WSSecurity Default
- Kerberos V5 HTTPS default
- SSL WSTransaction
- Username SecureConversation
- Username WSSecurity default
- WS-Addressing default
- WSHTTPS default
- WS-I RSP ND
- WS-ReliableMessaging persistent

The application server also provides other default policy sets that you can use or customize. To use the additional policy sets, you must import them from the default repository. Read about importing policy sets from the administrative console for more information.

The following default policy sets are provided:

WS-I RSP default

This policy set provides:
- Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
- Message integrity through digital signature that includes signing the body, time stamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
- Confidentiality through encryption that includes encrypting the body, signature elements, using the WS-SecureConversation and WS-Security specifications

LTPA WS-I RSP default

This policy set provides:
- Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
- Message integrity through digital signature that includes signing the body, time stamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications

- Confidentiality through encryption that includes encrypting the body, signature elements, using the WS-SecureConversation and WS-Security specifications
- A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service

Username WS-I RSP default

This policy set provides:

- Reliable message delivery to the intended receiver by enabling WS-ReliableMessaging
- Message integrity through digital signature that includes signing the body, time stamp, WS-Addressing headers and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
- Confidentiality through encryption that includes encrypting the body, signature elements, using the WS-SecureConversation and WS-Security specifications
- A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request

SecureConversation

This policy set provides:

- Message integrity through digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
- Message confidentiality through encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications

LTPA SecureConversation

This policy set provides:

- Message integrity through digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
- Message confidentiality through encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications
- A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service

Username SecureConversation

This policy set provides:

- Message integrity through digital signature that includes signing the body, time stamp, and WS-Addressing headers using WS-SecureConversation and WS-Security specifications
- Message confidentiality through encryption that includes encrypting the body, signature and signature confirmation elements, using WS-SecureConversation and WS-Security specifications
- A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request

WSAddressing default

Enables WS-Addressing support, which uses endpoint references and message addressing properties to facilitate the addressing of Web services in a standard and interoperable way.

WSHTTPS default

Provides SSL transport security for the HTTP protocol with Web services applications.

Kerberos V5 HTTPS default

This policy set provides message authentication with a Kerberos Version 5 token. Message integrity and confidentiality are provided by Secure Sockets Layer (SSL) transport security. This policy set follows the OASIS Kerberos Token Profile V1.1 and WS-Security specifications.

When you use this policy set, configure the basic authentication data and custom properties such as the com.ibm.wsspi.wssecurity.krbtoken.targetServiceName and

com.ibm.wsspi.wssecurity.krbtoken.targetServiceHost custom properties in the client bindings. For more information, see the Authentication generator or consumer token settings and Protection token settings (generator or consumer) topics.

WSReliableMessaging default

This policy set enables both WS-ReliableMessaging Version 1.1 and WS-Addressing and uses the minimum quality of service, *unmanaged non-persistent*. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent.

WSReliableMessaging persistent

This policy set enables both WS-ReliableMessaging and WS-Addressing and uses the maximum quality of service, *managed persistent*. This quality of service supports asynchronous Web service invocations and uses a service integration messaging engine and message store to manage the sequence state. Messages are processed within transactions, are persisted at the Web service requester server and at the Web service provider server, and are recoverable in the event of server failure. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent.

Because this policy set specifies managed persistent quality of service, you need to define bindings to the service integration bus and messaging engine that you want to use to manage the WS-ReliableMessaging state. For more information, see "Attaching and binding a WS-ReliableMessaging policy set to a Web service application using the administrative console" on page 657 or using the wsadmin tool.

WSReliableMessaging 1_0

This policy set enables both WS-ReliableMessaging Version 1.0 and WS-Addressing and uses the minimum quality of service, *unmanaged non-persistent*. This quality of service requires minimal configuration. However it is non-transactional and, although it allows for the resending of messages that are lost in the network, if a server becomes unavailable you will lose messages. In-order delivery is set to "false", so messages are not necessarily delivered in the order in which they were sent.

You can use this policy set with .NET-based Web services.

WSSecurity default

This policy set provides:

- Message integrity through digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications.
- Message confidentiality through encryption (using RSA public-key cryptography) to encrypt the body, signature and signature elements using WS-Security specifications.

LTPA WSSecurity default

This policy set provides:

- Message integrity through digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications.
- Message confidentiality through encryption (using RSA public-key cryptography) to encrypt the body, signature and signature elements using WS-Security specifications.
- A Lightweight Third Party Authentication (LTPA) token included in the request message to authenticate the client to the service.

Username WSSecurity default

This policy set provides:

- Message integrity through digital signature (using RSA public-key cryptography) to sign the body, time stamp, and WS-Addressing headers using WS-Security specifications.

- Message confidentiality through encryption (using RSA public-key cryptography) to encrypt the body, signature and signature elements using WS-Security specifications.
- A username token included in the request message to authenticate the client to the service. The username token is encrypted in the request.

WSTransaction

This policy set enables WS-Transaction, which provides:

- The ability to coordinate distributed transactional work atomically and interoperably using the WS-AtomicTransaction specification.
- The ability to coordinate loosely coupled business processes that are distributed across the heterogenous Web service environment, with the ability to compensate actions if a failure occurs in the business activity, using the WS-BusinessActivity specification.

SSL WSTransaction

This policy set enables WS-Transaction, which provides:

- The ability to coordinate distributed transactional work atomically, interoperably, and securely, using the WS-AtomicTransaction specification and SSL Transport security.
- The ability to coordinate loosely coupled business processes, with the ability to compensate actions if a failure occurs in the business activity, securely, using the WS-BusinessActivity specification and SSL Transport security.

Policy sets do not include environment or platform-specific information, such as keys for signing, keystore information, or persistent store information. This type of information is defined in the binding. A policy set attachment defines how a policy set is attached to service resources and bindings. The attachment definition is outside the policy set definition and is defined as meta-data associated with application data.

Bindings are made up of environment and platform-specific information. General bindings such as the service client or provider bindings for the global security domain can be shared across applications.

To enable policy sets to work with applications, bindings are needed. Use the administrative console to configure general bindings and application specific bindings. Read about defining binding information for policy sets for more information about working with attachments and bindings.

Related concepts

“WS-I RSP default policy sets” on page 809
The Reliable Asynchronous Message Profile (WS-I RSP) default policy sets are based on the Reliable Asynchronous Message Profile specification. The WS-I RSP default policy sets include the WS-I RSP default policy set, the Lightweight Third-Party Authentication (LTPA) WS-I RSP default policy set and the Username WS-I RSP default policy set. You can use these policy sets to simplify your Web services configuration.

“WS-ReliableMessaging default policy sets” on page 811
The WS-ReliableMessaging default policy sets are pre-configured to provide reliable message exchange between Web services. Two of these policy sets (WS-I RSP and WS-I RSP ND) are immediately available, and the rest are readily available for import from a default repository.

Web Services Addressing support
The Web Services Addressing (WS-Addressing) support in this product provides the environment for Web services that use the Worldwide Web Consortium (W3C) WS-Addressing specifications. This family of specifications provide transport-neutral mechanisms to address Web services and to facilitate end-to-end addressing.

“WSSecurity default policy sets” on page 813
The WSSecurity default policy sets are based on the Web Services Security (WS-Security) 1.0 and Web Services Addressing (WS-Addressing) specifications. The WSSecurity default policy sets include the WSSecurity default policy set, the Lightweight Third-Party Authentication (LTPA) WSSecurity policy set, and the Username WSSecurity policy set. Use the WSSecurity default policy sets to build secure Web services.

“SecureConversation default policy sets” on page 810
The SecureConversation default policy sets are based on the Web Services Secure Conversation Language (SecureConversation) standard that establishes a secure context, based on shared keys for the client and server to use for a series of messages. This standard provides a framework to define how to secure the message exchange across organizations. The SecureConversation default policy sets include the SecureConversation policy set, the Lightweight Third-Party Authentication (LTPA) SecureConversation policy set, and the Username SecureConversation policy set.

“WSHTTPS default policy set” on page 815
The WSHTTPS default policy set provides SSL transport security for the HTTP protocol with Web services applications.

“WSTransaction default policy sets” on page 814
The WSTransaction default policy sets are based on the WS-Transaction specification and provide transactional integrity for Web services. The WSTransaction default policy sets include the WSTransaction policy set and the SSL WSTransaction policy set.

Related tasks

“Defining and managing policy set bindings” on page 824
Policy set bindings contain platform specific information, like keystore, authentication information or persistent information, required by a policy set attachment. Use this task to create and manage bindings.

“Importing policy sets using the administrative console” on page 816
You can import predefined policy sets or import policy sets from a selected location using the administrative console.

“Defining and managing service client or provider bindings” on page 831
Service client or provider bindings are general bindings. You can create, copy, and manage general bindings such as the client or provider policy set bindings. These bindings provide system-specific configuration and can be reused across policy set attachments.

“Creating policy set attachments using the wsadmin tool” on page 515
Use the wsadmin tool, which supports the Jython and Jacl scripting languages, to define the policy set configuration for your Web services applications. You can attach policy sets to an application, Web service, endpoint, or specific operation.

Related reference

“Copy of default policy set and bindings settings” on page 815
Use this page to copy a policy set that you select from a list of available policy sets.

“Authentication generator or consumer token settings” on page 887
Authentication tokens are used to prove or assert an identity. Use the administrative console to add authentication token settings for message parts when you are editing a general binding.

“Protection token settings (generator or consumer)” on page 883
Use this page to configure protection tokens. Protection tokens sign messages to protect integrity or encrypt messages to provide confidentiality.

Web services specifications and APIs

Related information

WS-Policy working group

OASIS WS-SX Technical Committee

Overview of migrating policy sets and bindings

Policy sets are migrated during the product migration from Version 6.1 Feature Pack for Web Services to Version 7.0. This topic describes the different rules that apply to migrating policy sets and bindings.

For information about migrating the version of the product that you are running to Version 7.0, see migrating and coexisting.

Migration rules for policy sets

For the following policy sets, the migration code first checks for any application attachment for a policy set. If there is an application attachment, the migration code upgrades and renames the policy set to *oldPolicySet_wsfep*; then updates the policy set attachments from *oldPolicySet* to *oldPolicySet_wsfep* in Version 7.0. If there is no attachment for the policy set, then the policy set is not migrated or renamed to Version 7.0.

- LTPA RAMP default
- LTPA SecureConversation
- LTPA WSSecurity default
- Username RAMP default
- Username SecureConversation
- Username WSSecurity default

Other policy sets are migrated to Version 7.0 only if there is no other policy set in Version 7.0 with the same name.

Migration rules for bindings

In Version 7.0, custom bindings are called application specific bindings. Read Web services policy set bindings.

The following migration rules apply to application specific bindings, cell level default bindings, server level default bindings, and trust service bindings:

Application specific bindings

During migration, the Version 6.1 application specific bindings in the applications are not upgraded. They are copied with the applications to the Version 7.0 system. After migration, you can upgrade the Version 6.1 application specific bindings in an application using the administrative command, `upgradeBindings`.

Cell level default bindings

During migration, the cell level default bindings from the Version 6.1 Feature Pack for Web Services are copied to the Version 7.0 system, and they replace any existing Version 6.1 cell default bindings in the Version 7.0 system. The Version 6.1 cell default bindings are migrated to general bindings in Version 7.0.

Note:
There is a security fix for the Version 6.1 cell level default WSSecurity bindings. Run the script, *updateBindings.py* that is located in WAS_HOME/bin to fix issues with the Version 6.1 cell level default WSSecurity bindings.

You can install a WebSphere Application Server Version 6.1 Feature Pack for Web Services application within the WebSphere Application Server Version 7.0 environment. If your application contains Version 6.1 application specific bindings, the application is defined as a Version 6.1 application to the application server. Additionally, if your application is installed on a WebSphere Application Server Version 6.1 Feature Pack for Web Services server within the WebSphere Application Server Version 7.0 environment, the application specific binding is created as a Version 6.1 application specific binding.

When you create an application specific binding, the system checks to see if the application is installed on a Version 6.1 server or if the application contains any Version 6.1 application specific bindings. If either of these conditions is true, the system creates Version 6.1 application specific bindings. In addition, you are not allowed to assign a general binding to a policy set attachment for that application because general bindings are for Version 7.0 level of the product.

Server level default bindings

During migration, the server default bindings in the Version 6.1 Feature Pack for Web Services are copied over to the Version 7.0 server. The server default bindings are also migrated to general bindings. You can decide whether or not to use these general bindings as the Version 7.0 server default bindings. The Version 6.1 applications and other applications that are installed on a Version 6.1 server use the Version 6.1 default bindings.

Trust service bindings

Trust service bindings are migrated during the product migration. The migrated trust service bindings replace the trust service bindings in Version 7.0.

Installing and managing WSIF

An overview of where and how the Web Services Invocation Framework (WSIF) is installed as part of WebSphere Application Server, and a set of links to specific information about other aspects of managing your WSIF system.

About this task

WSIF is provided in a JAR file named `com.ibm.ws.runtime.jar`. The JAR file contains the core WSIF classes, and the Java, EJB, SOAP over HTTP and SOAP over JMS providers. Additional providers are packaged as separate JAR files.

When you install WebSphere Application Server, the `com.ibm.ws.runtime.jar` file is put on the WebSphere or Java virtual machine (JVM) class path.

WSIF requires no further configuration. WSIF is a thin abstraction layer between application code and the relevant invocation infrastructure.

For information about managing your WSIF system, see the following topics:

- wsif.properties file - Initial contents
- Enable security for WSIF

- Trace and log WSIF
- Troubleshoot the Web Services Invocation Framework
- WSIF messages
- WSIF - Known restrictions

wsif.properties file - Initial contents

The Web Services Invocation Framework (WSIF) properties are stored in the `com.ibm.ws.runtime.jar` file, in a properties file named `wsif.properties`. You might need to modify the contents of this file, for example to change the default SOAP provider, so for reference here is a copy of the “as shipped” contents of the `wsif.properties` file.

The `com.ibm.ws.runtime.jar` file is located in the *app_server_root*`/plugins` directory, where *app_server_root* is the root directory for your installation of IBM WebSphere Application Server.

You must keep the `wsif.properties` file on the class path, so that WSIF can find it and the client administrator can use it to configure WSIF. However if you make any changes to the file, you do not replace the original copy in the `com.ibm.ws.runtime.jar` file. Instead, you save the modified version in the *app_server_root*`/lib/properties` directory.

The following code is the initial contents of the `wsif.properties` file. All the possible properties are listed and described.

```
# Two properties are used to override which WSIFProvider is selected when there
# exists multiple providers supporting the same namespace URI. These properties are:
#
#    wsif.provider.default.CLASSNAME=N
#    wsif.provider.uri.M.CLASSNAME=URI
#
# CLASSNAME is the WSIFProvider class name
# N is the number of following default wsif.provider.uri.M.CLASSNAME properties
# M is a number from 1 to N to uniquely identify each wsif.provider.uri.M.CLASSNAME
#   property key.
# For example the following two properties would override the default SOAP provider
# to be the Apache SOAP provider:
#
# wsif.provider.default.org.apache.wsif.providers.soap.ApacheSOAP.WSIFDynamicProvider_ApacheSOAP=1
# wsif.provider.uri.1.org.apache.wsif.providers.soap.ApacheSOAP.WSIFDynamicProvider_ApacheSOAP=\
# http://schemas.xmlsoap.org/wsdl/soap/
#

# maximum number of milliseconds to wait for a response to a synchronous request.
# Default value if not defined is to wait forever.
# Timeout properties are only used by providers which support timeouts.
wsif.syncrequest.timeout=10000

# maximum number of seconds to wait for a response to an async request.
# if not defined on invalid defaults to no timeout
# Timeout properties are only used by providers which support timeouts.
wsif.asyncrequest.timeout=60
```

To enable your legacy Web services to continue to work with WSIF, you might need to change the default WSIF SOAP provider back to the former Apache SOAP provider.

Enabling security for WSIF

The steps to be taken to enable the Web Services Invocation Framework (WSIF) to interact with a security manager.

About this task

WSIF interacts with a security manager in the following ways:
- WSIF runs in the Java Platform, Enterprise Edition (Java EE) security context without modification.
- When WSIF is run under a Java EE container, port implementations can use the security context to pass on security tokens or credentials as necessary.
- WSIF implementations can automatically convert Java EE security context into appropriate context for onward services.

For WSIF to interact effectively with the WebSphere Application Server security manager, complete the following step:

To load the WSDL file, enable the **FilePermission** attribute in the `was.policy` file. This permission is required when a WSDL file is referred to using the `file://` protocol.

Web Services Invocation Framework troubleshooting tips

A set of specific tips to help you troubleshoot problems you experience with the Web Services Invocation Framework (WSIF).

For information about resolving WebSphere-level problems, see Diagnosing and fixing problems.

To identify and resolve Web Services Invocation Framework (WSIF)-related problems, you can use the standard WebSphere Application Server trace and logging facilities. If you encounter a problem that you think might be related to WSIF, you can check for error messages in the WebSphere Application Server administrative console, and in the application server `stdout.log` file. You can also enable the application server debug trace to provide a detailed exception dump.

A list of the WSIF run-time system messages, with details of what each message means, is provided in Message reference for WSIF.

A list of the main known restrictions that apply when using WSIF is provided in WSIF - Known restrictions.

Here is a checklist of major WSIF activities, with advice on common problems associated with each activity:

Create service

Handcrafted Web Services Description Language (WSDL) can cause numerous problems. To help ensure that your WSDL is valid, use a tool such as WebSphere Development Studio for iSeries (WDS) to create your Web service. For more details about creating services with WSDL in an iSeries environment, read the WSIF and WSDL chapter of *Developing and deploying applications* PDF book.

Define transport mechanism

For the Java Message Service (JMS), check that you have set up the Java Naming and Directory Interface (JNDI) correctly, and created the necessary connection factories and queues.

For SOAP, make sure that the deployment descriptor file `dds.xml` is correct - preferably by creating it using WebSphere Development Studio for iSeries (WDS) or similar tooling.

Create client - Java code

Follow the correct format for creating a WSIF service, port, operation and message. For examples of correct code, refer to the Address Book Sample in the *Developing and deploying applications* PDF book.

Compile code (client and service)

Check that the build path against code is correct, and that it contains the correct levels of JAR files.

Create a valid EAR file for your service in preparation for deployment to a Web server.

Deploy service

When you install and deploy the service EAR file, check carefully any messages given when the service is deployed.

Server setup and start

Make sure that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

WSIF setup

Check that the `wsif.properties` file is correctly set up. For more information about this file, refer to the Maintaining the WSIF properties file topic in the *Administering applications and their environment* PDF book.

Run client

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server launchClient tool.

Either check that you have defined the class path correctly to include references to your client classes, WSIF JAR files and any other necessary JAR files, or (preferably) run your client using the WebSphere Application Server launch client tool. For more information about this tool, refer to the Running application clients chapter of the *Developing and deploying applications* PDF book.

Here is a set of tips to help you troubleshoot commonly-experienced problems:

- "No class definition errors received when running client code."
- "Cannot find WSDL error."
- "Your Web service EAR file does not install correctly onto the application server." on page 972
- "There is a permissions problem or security error." on page 972
- "Using WSIF with multiple clients causes a SOAP parsing error." on page 972
- "JNDI lookup errors occur when using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines." on page 972
- "A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled." on page 973
- "The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not interoperate fully with services that are running on the former (Apache SOAP) provider." on page 973

"No class definition" errors received when running client code.

This problem usually indicates an error in the class path setup. Check that the relevant JAR files are included.

"Cannot find WSDL" error.

Some likely causes are:

- The application server is not running.
- The server location and port number in the WSDL are not correct.
- The WSDL is badly formed (check the error messages in the application server `stdout.log` file).
- The application server has not been restarted since the service was installed.

You might also try the following checks:

- Can you load the WSDL into your Web browser from the location specified in the error message?
- Can you load the corresponding WSDL binding files into your Web browser?

Your Web service EAR file does not install correctly onto the application server.

It is likely that the EAR file is badly formed. Verify the installation by completing the following steps:
- For an EJB binding, run the WebSphere Application Server tool `\bin\dumpnamespace`. This tool lists the current contents of the JNDI directory.
- For a SOAP over HTTP binding, open the `http://pathToServer/WebServiceName/admin/list.jsp` page (if you have the SOAP administration pages installed). This page lists all currently installed Web services.
- For a SOAP over JMS binding, complete the following checks:
  - Check that the queue manager is running.
  - Check that the necessary queues are defined.
  - Check the JNDI setup.
  - Use the "display context" option in the `jmsadmin` tool to list the current JNDI definitions.
  - Check that the Remote Procedure Call (RPC) router is running.

There is a permissions problem or security error.

Check that the WebSphere Application Server `server.policy` file (in the `/properties` directory) has the correct security settings. For more information about setting up security, see the Enabling security for WSIF topic in the *Securing applications and their environment* PDF book.

Using WSIF with multiple clients causes a SOAP parsing error.

Before you deploy a Web service to WebSphere Application Server, you must decide on the scope of the Web service. The deployment descriptor file `dds.xml` for the Web service includes the following line:

```
<isd:provider type="java" scope="Application" ......
```

You can set the `Scope` attribute to `Application` or `Session`. The default setting is `Application`, and this value is correct if each request to the Web service does not require objects to be maintained for longer than a single instance. If `Scope` is set to `Application` the objects are not available to another request during the execution of the single instance, and they are released on completion. If your Web service needs objects to be maintained for multiple requests, and to be unique within each request, you must set the scope to `Session`. If `Scope` is set to `Session`, the objects are not available to another request during the life of the session, and they are released on completion of the session. If scope is set to `Application` instead of `Session`, you might get the following SOAP error:

```
SOAPException: SOAP-ENV:ClientParsing error, response was:
FWK005 parse may not be called while parsing.;
nested exception is:

[SOAPException: faultCode=SOAP-ENV:Client; msg=Parsing error, response was:

FWK005 parse may not be called while parsing.;
        targetException=org.xml.sax.SAXException:
FWK005 parse may not be called while parsing.]
```

JNDI lookup errors occur when using the same names for JMS messaging queues and queue connection factories that run on application servers on different machines.

You should not use the same names for messaging queues and queue connection factories that run on application servers on different machines, because WSIF always looks first for JMS destinations locally, and only uses the full JNDI reference if it cannot find the destination locally. For example, if you run a Web service on a remote machine, and have an application server running locally that uses the same names for the messaging queues and queue connection factories, then WSIF will find and use the local queues even if the remote JNDI destination is provided in full in the WSDL service definition.

A JAX-RPC client running on WebSphere Application Server Version 5 uses SOAP over JMS to invoke a Web service running on a Version 5 application server. No user ID or password is required on the target MQ Series queue. After the application server is migrated to Version 6, and using Version 6 default messaging, client requests fail because basic authentication is now enabled.

The problem appears as a log message:

```
SibMessage W [:] CWSIT0009W: A client request failed in the application server
with endpoint <endpoint_name> in bus <your_bus> with reason: CWSIT0016E:
The user ID null failed authentication in bus <your_bus>.
```

For the steps to take to resolve the problem, see the following service integration technologies troubleshooting tip: “After you migrate an application server to WebSphere Application Server Version 6, existing Web services clients can no longer use SOAP over JMS to access services hosted on the migrated server”. This tip can be found in the Tips for troubleshooting service integration bus security section.

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not interoperate fully with services that are running on the former (Apache SOAP) provider.

This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. To enable interoperation, modify either your Web service or the WSIF default SOAP provider. For information about how to modify your provider, read the chapter “WSIF SOAP provider: working with legacy applications” in the *Developing and deploying applications PDF* book.

WSIF (Web Services Invocation Framework) messages

This topic contains a list of the WSIF run-time system messages, with details of what each message means.

WebSphere system messages are logged from a variety of sources, including application server components and applications. Messages logged by application server components and associated IBM products start with a unique message identifier that indicates the component or application that issued the message.

For more information about the message identifier format, see the topic Message reference.

WSIF0001E: An extension registry was not found for the element type “{0}”

Explanation: Parameters: {0} element type. No extension registry was found for the element type specified.

Action: Add the appropriate extension registry to the port factory in your code.

WSIF0002E: A failure occurred in loading WSDL from “{0}”

Explanation: Parameters: {0} location of the WSDL file. The WSDL file could not be found at the location specified or did not parse correctly

Action: Check that the location of the WSDL file is correct. Check that any network connections required are available. Check that the WSDL file contains valid WSDL.

WSIF0003W: An error occurred finding pluggable providers: {0}

Explanation: Parameters: {0} specific details about the error. There was a problem locating a WSIF pluggable provider using the J2SE 1.3 JAR file extensions to support service providers architecture. The WSIF trace file will contain the full exception details.

Action: Verify that a META-INF/services/org.apache.wsif.spi.WSIFProvider file exists in a provider jar, that each class referenced in the META-INF file exists in the class path, and that each class implements org.apache.wsif.spi.WSIFProvider. The class in error will be ignored and WSIF will continue locating other pluggable providers.

WSIF0004E: WSDL contains an operation type "{0}" which is not supported for "{1}"

Explanation: Parameters: {0} name of the operation type specified. {1} name of the portType for the operation. An operation type which is not supported has been specified in the WSDL.

Action: Remove any operations of the unsupported type from the WSDL. If the operation is required then make sure all messages have been correctly specified for the operation.

WSIF0005E: An error occurred when invoking the method "{1}" . ("{0}")

Explanation: Parameters: {0} name of communication type. For example EJB or Apache SOAP. {1} name of the method that failed. An error was encountered when invoking a method on the Web service using the communication shown in brackets.

Action: Check that the method exists on the Web service and that the correct parts have been added to the operation as described in the WSDL. Network problems might be a cause if the method is remote and so check any required connections.

WSIF0006W: Multiple WSIFProvider found supporting the same namespace URI "{0}" . Found ("{1}")

Explanation: Parameters: {0} the namespace URI. {1} a list of the WSIFProvider found.. There are multiple org.apache.wsif.spi.WSIFProvider classes in the service provider path that support the same namespace URI.

Action: A following WSIF0007I message will be issued notifying which WSIPFProvider will be used. Which WSIFProvider is chosen is based on settings in the wsif.properties file, or if not defined in the properties, the last WSIFProvider found will be used. See the wsif.properties file for more details on how to define which provider should be used to support a namespace URI.

WSIF0007I: Using WSIFProvider "{0}" for namespaceURI "{1}"

Explanation: Parameters: {0} the classname of the WSIFProvider being used. {1} the namespaceURI the provider will be used to support.. Either a previous WSIF0006W message has been issued or the SetDynamicWSIFProvider method has been used to override the provider used to support a namespaceURI.

Action: None. See also WSIF0006W.

WSIF0008W: WSIFDefaultCorrelationService removing correlator due to timeout. ID:"{0}"

Explanation: Parameters: {0} the ID of the correlator being removed from the correlation service. A stored correlator is being removed from the correlation service due to its timeout expiring.

Action: Determine why no response has been received for the asynchronous request within the timeout period. The wsif.asyncrequest.timeout property of the wsif.properties file defines the length of the timeout period.

WSIF0009I: Using correlation service - "{0}"

Explanation: Parameters: {0} the name of the correlation service being used. This identifies the name of the correlation service that will be used to process asynchronous requests.

Action: None. If a correlation service other than the default WSIF supplied one is required, ensure that it is correctly registered in the JNDI java:comp/wsif/WSIFCorrelationService namespace.

WSIF0010E: Exception thrown while processing asynchronous response - "{0}"

Explanation: Parameters: {0} the error message string of the exception. While processing the response from an executeRequestResponseAsync call an exception was thrown.

Action: Use the exception error message string to determine the cause of the error. The WSIF trace will have more details on the error including the exception stack trace.

WSIF0011I: Preferred port "{0}" was not available

Explanation: Parameters: {0} the user's preferred port. The preferred port set by the user on org.apache.wsif.WSIFService is not available

Action: None unless this message appears for long periods of time in which case the user might want to pick a different port as their preferred port.

WSIF - Known restrictions

This topic lists the main known restrictions that apply when using WSIF.

Threading

WSIF is not thread-safe.

External Standards

WSIF supports:

- SOAP Version 1.1 (not 1.2 or later).
- WSDL Version 1.1 (not 1.2 or later).

WSIF does not provide WS-I compliance, and it does not support the Java API for XML-based Remote Procedure Calls (JAX-RPC) Version 1.1 (or later).

Full schema parsing

WSIF does not support full schema parsing. For example, WSDL references in complex types in the schema are not handled, and attributes are not handled.

XML Schema “redefine” elements are not handled and are ignored.

SOAP WSIF does not support:

- SOAP headers that are passed as `<parts>`.
- Unreferenced attachments in SOAP responses, or the scenarios detailed in SOAP attachments - scenarios that are not supported.
- Document Encoded style SOAP messages.

  Note: This is not primarily a WSIF restriction. Although you can specify Document Encoded style in WSDL, it is not generally considered to be a valid option and is not supported by the Web Services Interoperability Organization (WS-I).

SOAP provider interoperability

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) does not fully interoperate with services that are running on the former (Apache SOAP) provider. This restriction is due to the fact that the IBM Web Service SOAP provider is designed to interoperate fully with a JAX-RPC compliant Web service, and Apache SOAP cannot provide such a service. For information on how to overcome this restriction, see WSIF SOAP provider: working with legacy applications.

WSIF’s support for SOAP faults is restricted to SOAP faults originating from a Web service that runs using the IBM Web Service SOAP provider.

Note: This is not primarily a WSIF restriction. The current SOAP faults specification does not prescribe how to encode a SOAP fault so that it maps to a Java exception. Consequently, each Web service run-time environment currently decides on its own SOAP fault format. The IBM Web Service SOAP provider can understand its own response SOAP faults, but not the SOAP faults from another provider.

Type mappings

The current WSIF default SOAP provider (the IBM Web Service SOAP provider) conforms to the JAX-RPC type mapping rules that were finalized after the former (Apache SOAP) provider was created. The majority of types are mapped the same way by both providers. The exceptions are: `xsd:date`, `xsd:dateTime`, `xsd:hexBinary` and `xsd:QName`. Both client and service need to use the same mapping rules if any of these four types are used. Below is a table detailing the mapping rules for these four types:

XML Data Type	Apache SOAP Java Mapping	JAX-RPC Java Mapping
xsd:date	java.util.Date	Not supported
xsd:dateTime	Not supported	java.util.Calendar
xsd:hexBinary	Hexadecimal string	byte [ ]
xsd:QName	org.apache.soap.util.xml.QName	javax.xml.namespace.QName

Arrays and complex types

WSIF does not support general complex types, it only handles complex types that map to Java

Beans. To use schema complex types, you must write your own custom serializers. The specific complex type and array support for WSIF outbound invocation of Web services is as follows:

- WSIF supports Java classes generated by WebSphere Studio Application Developer - Integration Edition (WSAD-IE) message generators (the normal case when WSDL files are downloaded from somewhere else). The WSAD-IE-based generation happens automatically when you use the BPEL editor, or the generation actions available on the Enterprise Services context menu, or the Business Integration toolbar.
- WSIF does not support Java beans generated by other tools, including the base WSAD tool.
- For WSAD-IE generated Java beans, attributes defined in the WSDL do not work. That is to say that these attributes, although they appear in the Java beans generated to represent the complex type, do not appear in the SOAP request created by WSIF.
- WSIF does not support arrays when they are a field of a Java bean. That is to say, WSIF only supports an array that is passed in as a named `<part>`. If an array is wrappered inside a Java bean, the array is not serialized in the same way.

Object Serialization
WSIF does not support serialization of objects across different releases.

Asynchronous invocation
WSIF supports synchronous invocation for all providers. For the JMS and the SOAP over JMS providers, WSIF also supports asynchronous invocation. You should call the `supportsAsync()` method before trying to execute an asynchronous operation.

The EJB provider
The target service of the WSIF EJB provider must be a remote-home interface, it cannot be an EJB local-home interface. In addition, the EJB stub classes must be available on the client class path.

Running outside WebSphere Application Server
WSIF is not supported for use outside WebSphere Application Server.

Trace and logging for WSIF

The Web Services Invocation Framework (WSIF) offers trace points at the opening and closing of ports, the invocation of services, and the responses from services. WSIF also includes a `SimpleLog` utility that can run trace when you are using WSIF outside of WebSphere Application Server.

About this task

If you want to enable trace for the WSIF API within WebSphere Application Server, and have trace, stdout and stderr for the application server written to a well-known location, see Enabling tracing and logging.

To trace the WSIF API, you need to specify the following trace string:

```
wsif=all=enabled
```

To enable the WSIF `SimpleLog` utility, through which you can run trace when using WSIF outside of WebSphere Application Server, complete the following steps:

1. Create a file named `commons-logging.properties` with the following contents:

```
org.apache.commons.logging.LogFactory=org.apache.commons.logging.impl.LogFactoryImpl
org.apache.commons.logging.Log=org.apache.commons.logging.impl.SimpleLog
```

2. Create a file named `simplelog.properties` with the following contents:

```
org.apache.commons.logging.simplelog.defaultlog=trace
org.apache.commons.logging.simplelog.showShortLogname=true
org.apache.commons.logging.simplelog.showdatetime=true
```

3. Put both these files, and the `commons-logging.jar` file, on the class path.

Results

The `SimpleLog` utility writes trace to the `System.err` file.

Getting started with the UDDI registry

This section covers the basic knowledge you need to get started, either as an administrator of a UDDI registry, or as a user of a UDDI registry that is already set up.

- If you are an administrator of a UDDI registry, that is, you are need to install (set up and deploy), customize, or manage a UDDI registry, use the following topics to get started.
  - UDDI registry Terminology introduces some terms that you need to understand to administer a UDDI registry
  - Setting up and deploying a new UDDI registry explains how to install a UDDI registry node by setting up the resources that it will use, and deploying the UDDI registry application.
  - Migrating the UDDI registry explains how to use the scripts provided to migrate UDDI registries from the previous version of WebSphere Application Server.
  - Migrating to Version 3 of the UDDI registry explains how to migrate UDDI Version 2 registries to UDDI Version 3, introduced in WebSphere Application Server Version 6.0, by creating a migration datasource.
  - Configuring UDDI registry security explains how to configure different levels of security for the UDDI registry, and how other UDDI node settings can influence security.
  - Managing the UDDI registry explains how to use the UDDI pages in the administrative console, or the UDDI registry Administrative interface, to administer a UDDI registry node. It also covers how to back up and restore your UDDI registry data.
  - UDDI node settings explains how to view and set UDDI properties and policies, and how to manage UDDI publishers, tiers, and user entitlements.
  - UDDI registry Management Interfaces covers details of programmatic interfaces that you can use to administer a UDDI registry node (UDDI registry Administrative (JMX) Interface), to add custom value set data to a UDDI registry (User Defined Value Set Support), and to export and import UDDI version 2 entities (UDDI Utility Tools).
  - “Removing and reinstalling the UDDI registry” on page 978
- If you are a user of a UDDI registry, use the following topics to get started.
  - UDDI registry Terminology introduces some terms that you might need to understand to use a UDDI registry.
  - Using the UDDI registry user interface explains how to access the UDDI User Console, which is a user interface that allows you to find UDDI entities and carry out simple UDDI publish operations.
  - UDDI registry SOAP Service End Points contains details for accessing the UDDI Version 3 inquiry, publish, security, and custody transfer APIs, as well as the UDDI Version 1 and Version 2 APIs.
  - UDDI registry Client Programming explains how to write UDDI client application programs. The recommended client programming interface is the UDDI Version 3 Client for Java.
  - JAXR Provider for the UDDI registry is for users who want to use the Java API for XML Registries (JAXR) to access UDDI.
  - User Defined Value Set Support in the UDDI registry explains how to add custom value set data to a UDDI registry.
- For either role, UDDI registry troubleshooting might be useful if you encounter any problems or unexpected behavior while using the UDDI registry, and Messages links to messages that you might see.

Removing and reinstalling the UDDI registry

This task describes the points to consider when you remove a UDDI application, remove a UDDI registry node, or reinstall a UDDI registry node, and describes how to achieve this.

About this task

A UDDI registry node consists of an enterprise application, a store of data (using a relational database management system) referred to as the UDDI database, and a means to connect the application to the data (a data source and related elements). All the data relating to UDDI is stored within the UDDI database and therefore exists irrespective of the UDDI application.

With these facts in mind, consider the following options:

- To remove a UDDI registry node from a WebSphere Application Server, you do not need to delete the database. You just need to delete the UDDI application and any associated resources, such as the data source used (and J2EE Connector Architecture authentication data if used), because the data in the UDDI database is separate from the UDDI application.
- To start a new UDDI registry node, you do not need to remove the UDDI application. Instead, you can create a new replacement node by changing the data source that the UDDI application uses to access the new UDDI database.
- To remove a UDDI application or to remove a UDDI registry node, see Removing a UDDI Node. Remove the UDDI application if you no longer want a UDDI facility on a particular application server. You can subsequently move the UDDI registry node to a different application server.
- To reinstall a UDDI registry node, see Reinstalling the UDDI application. Reinstall the UDDI application if you wish to continue to provide the UDDI facility on a particular application server.

Removing a UDDI registry node

You can remove a UDDI registry node completely, or just remove the UDDI registry application, delete the UDDI registry database, or move a UDDI registry to another server or profile.

About this task

To remove a UDDI registry node completely, you need to remove the UDDI registry application and delete the UDDI registry database. However, there might be situations where you want to perform only one of these tasks:

Removing the UDDI registry application from an application server.
You might want to do this to reinstall the application, either because it has become corrupted, or to apply service. See also Reinstalling the UDDI registry application.

Deleting the UDDI registry database
You might want to do this to use a different database product as the persistence store for your UDDI data, to delete all your UDDI registry data to publish fresh data (for example, if you have completed a test cycle), or to re-initialize the UDDI registry node with new UDDI property settings (for example, to move from a default UDDI node to a customized UDDI node). Note that if you delete a UDDI registry database, all UDDI data for that UDDI registry is lost.

Moving a UDDI registry to another server or profile
You might want to do this after you create a new profile and you want to move the UDDI registry to the new profile.

Completely removing a UDDI registry node from an application server
You might want to do this to remove a UDDI registry that was used for testing.

Depending on what you wish to achieve, complete **one** of the following steps:

1. **Removing a UDDI registry application**

Start a Qshell session. To do this, enter the STRQSH command from the i5/OS command line.

To remove the UDDI application from an application server, run the wsadmin script uddiRemove.jacl, as shown, from the *app_server_root*/bin directory.

The syntax of the command is as follows:

```
wsadmin [-profileName profile_name] -f uddiRemove.jacl
               node_name
               server_name
               [default]
```

where

- '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile is used.
- *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).
- 'default' is optional and is applicable only for Apache Derby, and then only if the default option was used when the uddiDeploy.jacl script was run to deploy the UDDI registry. Specifying default will remove the UDDI Apache Derby datasource but **not** the UDDI Apache Derby database.

By default, output appears on screen. To direct the output to a log file, add '> *removeuddi.log*' (where removeuddi.log can be any log name of your choice) to the end of the command.

For example, to remove the UDDI application from server 'server1' running in node 'MyNode' and send any messages to the file removeuddi.log:

```
wsadmin -profileName myProfile -f uddiRemove.jacl MyNode server1 > removeuddi.log
```

Note: You can also remove the UDDI registry application using the administrative console in the usual way, by selecting the application in the **Enterprise Applications** view and clicking **Uninstall**.

2. **Deleting a UDDI registry database**

   This action destroys all UDDI data in that UDDI registry.

   a. Stop the server that is hosting the UDDI registry application.
   b. Delete the database:
      - For DB2 for iSeries, use either iSeries Navigator or a 5250 Session to delete the schemas IBMUDI30 and IBMUDS30.
      - For Oracle, delete the schemas IBMUDDI, IBMUDI30 and IBMUDS30.
      - For Apache Derby, delete the directory tree containing the UDDI database. By default, this directory tree is located in *profile_root*/databases/com.ibm.uddi/UDDI30.

3. **Moving a UDDI registry to another server or profile**

   a. Make sure that the UDDI database will still be accessible. For example, if you are using a remote database, make sure that the new server can access it. If your database will not be accessible, or it will be deleted after the move, copy it to a new, accessible location. For example, if you want to move the UDDI registry to a new profile and then delete the old profile, any databases stored in the profile (such as an Apache Derby database created as part of the creation of a default UDDI node) will also be deleted.
   b. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.
   c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted. If the default option was not valid for your configuration, delete the following objects:
      - the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
      - any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
      - any J2C Authentication Data Entry that was created.

d. In the new server create a J2C authentication data entry (if appropriate), JDBC provider and datasource, to reference the existing database (for more information see the relevant steps in Setting up a customized UDDI node).

e. Deploy the UDDI registry application as described in Deploying the UDDI registry application, noting that you should not specify the default option even if you previously used this option to set up a default UDDI node. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.

Note: The UDDI node name will remain the same as before. If the UDDI node name included the node name and server name of the original server, there will be a mismatch between the UDDI node name and the node name and server name of the new server. However this name mismatch will not affect the functioning of the UDDI registry node.

f. Check that the UDDI data can be accessed. If you are using a copy of the original UDDI registry database, you can now delete the original as described above.

4. **Completely removing a UDDI registry node**

To completely remove a UDDI registry node from an application server, you need to remove the UDDI registry application and database, and also the resources that were used to reference the UDDI registry database.

a. Remove the UDDI registry application by running the uddiRemove.jacl script as described above.

b. Delete the UDDI registry database, as described above.

c. If you ran the uddiRemove.jacl script using the default option, the datasource and related objects have already been deleted so no further action is required. If the default option was not valid for your configuration, delete the following objects:

- the UDDI datasource that references the UDDI registry database (this was created when you set up the UDDI registry).
- any UDDI JDBC provider that was created (if you did not reuse an existing JDBC provider).
- any J2C Authentication Data Entry that was created.

Reinstalling the UDDI registry application

You can reinstall the UDDI registry application to remove the existing UDDI application and reinstall it with the existing UDDI database.

About this task

Perform this task if you wish to change the UDDI application code, but you want to continue to provide UDDI services with your existing UDDI database.

1. Make a note of any changes that you made to the installed UDDI application that you wish to keep, for example changes to security role mappings, changes to the deployment descriptor (web.xml) in v3soap.war, v3gui.war, v3soap.war, or soap.war, or customization of the UDDI user interface (GUI). All such changes are lost during the reinstallation process; if there are changes that you wish to keep, you must reapply them later.
2. Run the wsadmin script uddiDeploy.jacl from the *app_server_root*/bin directory, as shown. Note the following points:
   - Do not specify the default option, even if you used this option previously to set up a default UDDI node. If you use the default option, you might encounter an error during deployment, or in some circumstances your existing UDDI data might be overwritten.

   At a command prompt, enter:

```
wsadmin [-conntype none] [-profileName profile_name] –f uddiDeploy.jacl
               node_name
               server_name
```

   where

- '-profileName *profile_name*' is optional, and is the name of the profile in which the UDDI application is deployed. If you do not specify a profile, the default profile is used.
- '-conntype none' is optional, and is needed only if the application server is not running.
- *node_name* and *server_name* are the names of the WebSphere node and application server in which the UDDI application is deployed (these are the names that you specified when you ran uddiDeploy.jacl to install the UDDI application).

Optionally, you can redirect the output from this command to a log file. To do this, add '> *log_name*.log' at the end of the command, where *log_name*.log is the name of the log file to create.

The command removes the existing UDDI application and reinstalls it.

Note: This procedure does not change your existing JDBC provider, datasource and any J2C authdata entry. Your existing UDDI registry data, including UDDI entities as well as property and policy settings, are also unaffected.

3. If you noted any changes in step 1, reapply them now.
4. For the reapplied changes to take effect, start or restart the application server.

Creating a DB2 distributed database for the UDDI registry

Perform this task if you want to use DB2 on the Windows, Linux or UNIX operating systems, as the database store for your UDDI registry data.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

<DataBaseName>
The name of the UDDI registry database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

<DB2UserID>
A DB2 userid with administrative privileges.

<DB2Password>
The password for the DB2 userid.

<BufferPoolName>
The name of a buffer pool to be used by the UDDI registry database. A suggested name is uddibp, but any name can be used, because the buffer pool is created as part of this task.

<TableSpaceName>
The name of a table space. A suggested value is uddits, but any name can be used.

<TempTableSpaceName>
The name of a temporary table space. A suggested value is udditstemp, but any name can be used, because the temporary table space is created as part of this task.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

1. Change directory to *app_server_root*/UDDIReg/databaseScripts.
2. Start the DB2 Command Line Processor by entering `db2` at the command prompt.
3. Run the following command to setup the DB2 environment variables:

```
set DB2CODEPAGE=1208
```

4. Create the DB2 database by entering the following command:
   ```
   create database <DataBaseName> using codeset UTF-8 territory en
   ```
   where <DataBaseName> is the name of the database being created.
5. Configure the DB2 database by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `update db cfg for <DataBaseName> using applheapsz 2048`
   c. `update db cfg for <DataBaseName> using logfilsiz 8192`
   d. `connect reset`
   e. `terminate`
6. Create additional database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create bufferpool <BufferPoolName> size 250 pagesize 32K`
   c. `connect reset`
   d. `terminate`
   e. `force application all`
   f. `terminate`
   g. `stop`
   h. `start`
7. Create further database structures by entering the following commands:
   a. `connect to <DataBaseName> user <DB2UserID> using <DB2Password>`
   b. `create regular tablespace uddits pagesize 32K managed by system using ('<TableSpaceName>') extentsize 64 prefetchsize 32 bufferpool <BufferPoolName>`
   c. `create system temporary tablespace <TempTableSpacename> pagesize 32K managed by system using ('<TempTableSpacename>') extentsize 32 overhead 14.06 prefetchsize 32 transferrate 0.33 bufferpool <BufferPoolName>`
8. Exit the DB2 Command Line Processor and enter the following commands exactly as shown, noting that one step uses -vf rather than -tvf (on Windows platforms, run the commands from the db2cmd window). These commands define the database structures needed to store the UDDI data:
   a. `db2 -tvf uddi30crt_10_prereq_db2.sql`
   b. `db2 -tvf uddi30crt_20_tables_generic.sql`
   c. `db2 -tvf uddi30crt_25_tables_db2udb.sql`
   d. `db2 -tvf uddi30crt_30_constraints_generic.sql`
   e. `db2 -tvf uddi30crt_35_constraints_db2udb.sql`
   f. `db2 -tvf uddi30crt_40_views_generic.sql`
   g. `db2 -tvf uddi30crt_45_views_db2udb.sql`
   h. `db2 -vf uddi30crt_50_triggers_db2udb.sql`
   i. `db2 -tvf uddi30crt_60_insert_initial_static_data.sql`
9. [Optional] Enter the following command if you want the database to be used as a default UDDI node:
   ```
   db2 -tvf uddi30crt_70_insert_default_database_indicator.sql
   ```

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating a DB2 for iSeries database for the UDDI registry

Perform this task if you want to use DB2 for iSeries as the database store for your UDDI registry data.

Before you begin

IBMUDI30 and IBMUDS30 are the default names of the UDDI registry schema in the SQL scripts mentioned below. These default names are the recommended values and are assumed throughout the UDDI documentation. If you want to use different names, modify the SQL files listed below, then substitute your own names whenever you see 'IBMUDI30' or 'IBMUDS30' in other sections of the documentation.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Use iSeries Navigator to run SQL scripts.
   a. Open iSeries Navigator.
   b. Expand **My Connections** → **iSeriesName** → **Databases**.
   c. Select **iSeriesName**.
   d. Right-click **Run SQL Scripts...**.

   A **Run SQL Scripts** window opens.
2. Open the i5/OS DB2 SQL files.
   a. Map a network drive to the root directory of your i5/OS server's integrated file system.
   b. In Windows Explorer, expand the `WAS_HOME/UDDIReg/databaseScripts` directory.
   c. Open the following SQL files with a text editor (such as Notepad):
      - uddi30crt_10_prereq_db2_iSeries.sql
      - uddi30crt_20_tables_generic_iSeries.sql
      - uddi30crt_25_tables_db2udb_iSeries.sql
      - uddi30crt_30_constraints_generic_iSeries.sql
      - uddi30crt_35_constraints_db2udb_iSeries.sql
      - uddi30crt_40_views_generic_iSeries.sql
      - uddi30crt_45_views_db2udb_iSeries.sql
      - uddi30crt_50_triggers_db2udb_iSeries.sql
      - uddi30crt_60_insert_initial_static_data_iSeries.sql
3. Copy the text to the **Run SQL Scripts** window.
   a. In the text editor of the file `uddi30crt_10_prereq_db2_iSeries.sql`, click **Edit** → **Select All**.
   b. Click **Edit** → **Copy**.
   c. In the **Run SQL Scripts** window, click **Edit** → **Paste**.
   d. Click **Run** → **All**.
   e. After the script completes running, select all the SQL text and delete it from the **Run SQL Scripts** window.
   f. Repeat the above steps for all the SQL scripts listed in step 2.

   Note: iSeries Navigator for i5/OS V5R2 will encounter an error on the last line of the SQL script when running the `uddi30crt_25_tables_db2udb_iSeries.sql` script. This error can be ignored. i5/OS V5R2 does support running UDDI in a clustered high availability environment. For more information, see the WebSphere Application Server for i5/OS tech notes.
4. This step should only be run if you want your database to be used as a default UDDI node.
   a. Open `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 2 above.
   b. Copy and run `uddi30crt_70_insert_default_database_indicator.sql` as described in Step 3 above.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Apache Derby database for the UDDI registry

Perform this task to use Apache Derby (embedded or network) as the database store (either local or remote) for your UDDI registry.

Before you begin

The following steps use a number of variables. Before you start, decide appropriate values to use for these variables. The variables, and suggested values, are:

arg1 The path of the SQL files. On a standard installation, this is *app_server_root*/UDDIReg/databasescripts

arg2 The path to the location where you want to install the Apache Derby database.

For example, *profile_root*/databases/com.ibm.uddi

arg3 The name of the Apache Derby database. A recommended value is UDDI30, and this name is assumed throughout the UDDI information. If you use another name, substitute that name when UDDI30 is used in the information center.

arg4 An optional argument, which must either be the string 'DEFAULT', or be omitted. Specify DEFAULT if you want the database to be used as a default UDDI node. This argument is case sensitive.

If you want to create a remote database, refer first to the database product documentation about the relevant capabilities of the product.

About this task

You need to perform this task only once for each UDDI registry, as part of setting up and deploying a UDDI registry.

1. Start a Qshell session by entering the STRQSH command from the i5/OS command line.
2. Run the following Java -jar command from the *app_server_root*/UDDIReg/databaseScripts directory, to create a UDDI Apache Derby database using UDDIDerbyCreate.jar.

   ```
   java -Djava.ext.dirs=app_server_root/derby/lib:app_server_root/java/jre/lib/ext -jar UDDIDerbyCreate.jar
   arg1 arg2 arg3 arg4
   ```

   If the Apache Derby database already exists, you are asked if you want to recreate it. If you choose to recreate the database, your existing database is deleted and a new one is created in its place. If you choose not to recreate the database, the command exits and a new database is not created.

   Note: If the application server has already accessed the existing Apache Derby database, the uddiDeploy.jacl script cannot recreate the database. Use the uddiRemove.jacl script to remove the database, as described in "Removing a UDDI registry node" on page 978, restart the server, and run the uddiDeploy.jacl script again.
3. If you are using a remote database (which requires network Apache Derby), or you want to use network Apache Derby for other reasons, for example, if you want to use Apache Derby with a cluster, configure the Apache Derby Network Server framework. For details, see the section about managing the Derby Network Server in the Derby Server and Administration Guide.

What to do next

Continue with setting up and deploying your UDDI registry node.

Creating an Oracle database for the UDDI registry

Perform this task if you want to use Oracle as the database store for your UDDI registry data. You need only do this once for each UDDI registry, as part of setting up and deploying a UDDI registry.

About this task

The supported versions of Oracle are Version 9i[7] and Version 10g[8]

You can create a remote database using Oracle, but not a local database. Because the database is remote, use the database product documentation to familiarize yourself with the relevant capabilities of the product before proceeding with this task.

Before you begin

This task creates three new schemas: ibmuddi, ibmudi30 and ibmuds30. You will be unable to complete this task if you already have existing schemas with these names.

The steps below use a number of variables for which you need to enter appropriate values. You should decide the values that you will use before you start. The variables used are:

<OracleUserID>
is the Oracle userid to be used to create the database.

<OraclePassword>
is the password for the Oracle userid.

1. Run the following commands:
   a. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_10_prereq_oracle.sql`
   b. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_20_tables_generic.sql`
   c. Complete one of the following actions depending on your level of Oracle:
      - For Oracle 9i:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle_pre10g.sql
```

      - For Version 10g and later:

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_25_tables_oracle.sql
```

   d. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_30_constraints_generic.sql`
   e. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_35_constraints_oracle.sql`
   f. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_40_views_generic.sql`
   g. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_45_views_oracle.sql`
   h. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_50_triggers_oracle.sql`
   i. `sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_60_insert_initial_static_data.sql`

7.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters; instanceParms (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 8192 characters; overviewURL (tModelInstanceInfo) maximum 4000 bytes, UDDI specification 4096 characters; Digital Signature maximum 4000 bytes.

8.

Restrictions:
discoveryURL (Business) maximum 4000 bytes, UDDI specification 4096 characters; accessPoint (bindingTemplate) maximum 4000 bytes, UDDI Specification 4096 characters.

2. This last command should only be run if you want the database to be used as a default UDDI node.

```
sqlplus <OracleUserID>/<OraclePassword> @ uddi30crt_70_insert_default_database_indicator.sql
```

What to do next

Continue with setting up and deploying your UDDI registry node.

Applying an upgrade to the UDDI registry

Use this task to apply an iFix, Fix Pack or Refresh Pack to the UDDI registry.

About this task

Note: Some upgrades may require additional steps. Before you complete this task, refer to the readme file for the upgrade.

1. Apply the WebSphere Application Server iFix, Fix Pack or Refresh Pack to your application server, or servers, using the WebSphere Application Server Update Installer. You must repeat this process for each server on which you want to incorporate the UDDI upgrade.
2. If you have not yet deployed a UDDI registry, no further action is required, because updates to the UDDI registry takes effect when you first deploy UDDI into any of your application server profiles.
3. If you have already deployed a UDDI registry to one or more application server profiles, to apply the upgrade, redeploy the UDDI application, as described in Reinstalling the UDDI registry application. The existing UDDI application is removed and the updated application is deployed.

Configuring SOAP API and GUI services for the UDDI registry

For SOAP application programming interface (API) and graphical user interface (GUI) UDDI services, you can configure whether or not the API is secure. For SOAP API services you can also configure the default pool size.

Before you begin

You must have installed the UDDI registry application.

About this task

For the Version 1 and Version 2 SOAP interface, you can configure the following properties:

- *defaultPoolSize*. This is the number of SOAP parsers with which to initialize the parser pool for the SOAP interface. You can set this independently for the Publish (uddipublish) and Inquiry (uddi) APIs. For example, if you expect more inquiries than publish requests through the SOAP interface, you can set a larger pool size for the Inquiry API. The default initial size for both APIs is 10.
- Whether the API is to be secure (accessed using HTTPS) or insecure (accessed using HTTP). The default for Publish is to use HTTPS and Inquiry to use HTTP.

For the Version 3 SOAP interface, you can specify whether Publish, Custody Transfer, Security and Inquiry APIs are to be secure (accessed using HTTPS) or insecure (accessed using HTTP).

For the Version 3 GUI interface, you can specify whether the Publish and Inquiry API services are to be secure (accessed using HTTPS) or insecure (accessed using HTTP).

- Optional: Configure Version 1 and Version 2 SOAP API services by performing the following steps.
  1. Edit the active deployment descriptor (web.xml) for the Version 1 and Version 2 SOAP module (soap.war).

     This file is located in the following directory:

profile_root/config/cells/*cell_name*/applications/
UDDIRegistry.*node_name*.*server_name*.ear/deployments/
UDDIRegistry.*node_name*.*server_name*/soap.war/WEB-INF

2. To modify defaultPoolSize for Version 1 or Version 2 Publish, modify the 'param-value' element in the servlet with 'servlet-name' = uddipublish
3. To modify defaultPoolSize for Version 1 or Version 2 Inquiry, modify the 'param-value' element in the servlet with 'servlet-name' = uddi
4. To modify the user data constraint transport guarantee for Version 1 or Version 2 publish, which determines whether the Publish service is to be confidential (accessed using HTTPS) or insecure (using HTTP), find the 'security-constraint' with id = 'UDDIPublishTransportConstraint' and set its 'user-data-constraint' 'transport-guarantee' to CONFIDENTIAL or NONE.
5. Stop and restart the application server for the changes to take effect.

- Optional: For the Version 3 SOAP interface, specify whether the Publish, Custody Transfer, Security and Inquiry API services are to be secure or insecure by performing the following steps. The defaults are for Publish, Custody Transfer and Security APIs to use HTTPS, and for the Inquiry API to use HTTP.
  1. Edit the active deployment descriptor (web.xml) for the Version 3 SOAP module (v3soap.war).

     This file is located in the following directory:

     profile_root/config/cells/*cell_name*/applications/
     UDDIRegistry.*node_name*.*server_name*.ear/deployments/
     UDDIRegistry.*node_name*.*server_name*/v3soap.war/WEB-INF
  2. Locate the <security-constraint> element for the service that you want to modify by searching the deployment descriptor (web.xml) file for the relevant <display-name> value (see the table below). When you have found the <security-constraint> element, set its <user-data-constraint> <transport-guarantee> to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP). The <security-constraint> element represents a type of service, and contains the <display-name> element, <user-data-constraint> <transport-guarantee> elements, and other elements.

Type of UDDI service	Value of <security-constraint> <display-name> element
Publish	AxisServlet Publish Resource Collection
Custody transfer	AxisServlet CustodyTransfer Resource Collection
Security	AxisServlet Security Resource Collection
Inquiry	AxisServlet Inquiry Resource Collection

  3. Stop and restart the application server for the changes to take effect.
- Optional: For the Version 3 GUI interface, specify whether the Publish and Inquiry API services are to be secure or insecure by performing the following steps. The defaults are for the Publish API to use HTTPS, and the Inquiry API to use HTTP.
  1. Edit the active deployment descriptor (web.xml) for the Version 3 GUI module (v3gui.war).

     This file is located in the following directory:

     profile_root/config/cells/*cell_name*/applications/
     UDDIRegistry.*node_name*.*server_name*.ear/deployments/
     UDDIRegistry.*node_name*.*server_name*/v3gui.war/WEB-INF
  2. Use the table below to find the id value for the <user-data-constraint> element that corresponds with the service that you want to change. Search for this value in the deployment descriptor (web.xml) file to find the <user-data-constraint> element, then set the <user-data-constraint> <transport-guarantee> to either CONFIDENTIAL (the service can only be accessed using HTTPS) or NONE (the service can be accessed using HTTP).

Type of UDDI service	Value of <user-data-constraint id>
Publish	UDDIPublishTransportConstraint
Inquiry	UDDIInquireTransportConstraint

3. Stop and restart the application server for the changes to take effect.

Managing the UDDI registry

You can use either the WebSphere Application Server administrative console or the Java Management Extensions (JMX) management interface to manage UDDI Registries.

About this task

In previous versions of WebSphere Application Server and the UDDI registry, a properties file was used, but from WebSphere Application Server Version 6, you manage all the policies and properties of the UDDI registry through either the JMX management interface or the administrative console.

JMX can be used to monitor and configure UDDI registries programmatically, and is explained in Using administrative programs (JMX). See UDDI registry Administrative (JMX) Interface for full details on using the UDDI administrative interface. To manage UDDI registries using the WebSphere Application Server administrative console, start from the UDDI link in the left navigation pane as described below.

Using the UDDI management functions available in the WebSphere Application Server administrative console, you can perform the following operations:

- view and manage the status of all UDDI nodes in a cell
- initialize UDDI nodes with required settings
- configure general properties that affect UDDI runtime behavior
- manage UDDI policy settings
- create, view and update UDDI publishers
- create, view and update publisher tiers which limit how many UDDI entries may be published
- view and manage the status of value sets

If WebSphere Application Server security is enabled, you will need to log in to the WebSphere Administrative Console, supplying a valid userid and password, in order to use the UDDI management functions.

UDDI node collection

Use this page to manage the UDDI nodes in this cell. Each UDDI node represents an individual UDDI registry application. A UDDI node will only appear in this list if its underlying UDDI application is started. The status of the UDDI node indicates whether the node is activated (available to accept API requests), deactivated (not allowing user requests), or not initialized. UDDI nodes that are not initialized require some properties to be set before they can be initialized and activated.

To view this administrative console page, click **UDDI** → **UDDI Nodes**.

Each UDDI node is represented by a UDDI Node ID, Description, UDDI Application Location and Status. To manage an individual UDDI node, click on its UDDI Node ID link. This takes you to the Configuration page where you can manage its general properties, initialize it if the status is set to *Initialization Pending*, and access pages for managing policies, UDDI publishers, tiers and value sets.

UDDI Node ID

Specifies the identifier for the UDDI node.

Description

Specifies the description of the UDDI node.

UDDI Application Location

Specifies the server in which the UDDI registry application is deployed and running.

Status
Specifies the status of the UDDI node.

The Status of the UDDI node can be *Not initialized*, *Initialization pending*, *Initialization in progress*, *Migration in progress*, *Migration pending*, *Value set creation in progress*, *Value set creation pending*, *Activated* or *Deactivated*. If a node is in the *Initialization pending* state, you must initialize it before you can activate it. If you attempt to initialize the node and it remains in a pending state, an error occurred during migration or initialization.

To activate UDDI nodes that are *Deactivated*, select them by checking the corresponding check boxes in the Select column and click **Activate**. Similarly to deactivate UDDI nodes, select them and click **Deactivate**.

Note: Restarting the UDDI application, or the application server, will always result in the reactivation of the UDDI node, even if the node was previously deactivated.

UDDI node settings
This topic contains details of the general properties that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

By clicking on a node in the UDDI node ID column the UDDI node detail page is displayed. The UDDI node detail page displays a set of General Properties for the UDDI node, some of which may be editable depending on the state of the node. There are also links to Additional Properties (Value sets, Tiers and UDDI Publishers) and links to Policy Groups where you can view and change UDDI node policy.

Unless you installed the UDDI node as a default UDDI node (as defined in UDDI registry terminology), there are some important general properties that you must set before you can initialize the UDDI node. These properties are marked as being required (indicated by the presence of a '*' next to the input field). You may set the values as many times as you wish before initialization. However, after you initialize the UDDI node, these properties will become read only for the lifetime of that UDDI node. It is very important to set these properties correctly. You can set other general properties of the UDDI node before, and after, initialization.

After you set the general properties to appropriate values, click **OK** to save your changes and exit the page, or **Apply** to save your changes and remain on the same page. At this point the changes are stored.

If the UDDI node is in the *Not Initialized* state, indicated on the UDDI node detail page by the presence of an **Initialize** button (above the General Properties section), you can initialize UDDI node by clicking **Initialize**. This operation can take a while to complete. It is important to save any changes you made to the general properties by clicking **Apply** or **OK**, before you click **Initialize**.

UDDI Node ID:

Specifies the unique identifier that is given to a UDDI node in a UDDI registry. The node ID must be a valid UDDI key. The value will also be the domain key for the UDDI node.

The value for the node ID will also be the domain key for this UDDI node.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:node_id

UDDI node description:

Specifies the description of this UDDI node.

Required	Yes
Data type	String
Default	WebSphere UDDI registry default node

Root key generator:

Specifies the root key space of the registry. Registries intending to become affiliate registries might want to specify a root key space in a partition below the root key generator of the parent root registry, for example, uddi:thisregistry.com:keygenerator.

Required	Yes
Data type	String
Default	uddi:*cell_name*:*node_name*:*server_name*:keyspace_id:keygenerator

Prefix for generated discoveryURLs:

Specifies the URL prefix that is applied to generated discoveryURLs in businessEntity elements, so they can be returned on HTTP GET requests. This property applies to UDDI version 2 API requests only. Set this prefix to a valid URL for your configuration, and do not change it unless absolutely necessary.

The format is `http://hostname:port/uddisoap/`, where `uddisoap` is the context root of the UDDI version 2 SOAP servlet.

Although this field is not required, you should set it so that the required and valid URL is generated in response to version 2 GET requests. After you have set the prefix you should not alter it unless a subsequent configuration change renders it invalid; if you change the prefix, discoveryURLs generated using the earlier prefix will no longer work.

Required	No
Data type	String
Default	http://localhost:9080/uddisoap

Host name for UDDI node services:

Specifies the host name root used by the UDDI node to model API services in its own node business entity. This value must be the fully qualified domain name, or IP address, of the network host.

The UDDI node provides Web services that implement each of the UDDI API sets that it supports. The host name is used to generate access point URLs in the bindingTemplate elements for each of the services. The access point URL is generated by prefixing the host name value with a protocol, such as http, and suffixing it with the corresponding host port number. The access point URL must resolve to a valid URL.

Data type	String
Default	localhost

Host HTTP port:

Specifies the port number used to access UDDI node services with HTTP. This port number must match the WebSphere Application Server port for HTTP requests.

Data type	Integer

Default 9080

Host HTTPS port:

Specifies the port number used to access UDDI node services with HTTPS. This port number must match the WebSphere Application Server port for HTTPS requests.

Data type Integer
Default 9443

Maximum inquiry result set size:

Specifies the maximum size of result set which the registry will process for an inquiry API request.

If the result set exceeds this value, an E_resultSetTooLarge error is returned to the user. If you set this value too low, and users use imprecise search criteria, the likelihood of receiving an E_resultSetTooLarge error is increased. Setting the value higher allows larger result sets but can cause increased response times.

Data type Integer
Default 500
Range 0 to 1024

Maximum inquiry response set size:

Specifies, for inquiry API requests, the maximum number of results returned in each response. Do not set this value higher than the value for **Maximum inquiry results**.

If the result set contains more results than this value, the response will include only a subset of those results. The user can retrieve the remaining results using the listDescription feature as described in the UDDI specification. If you set this value too low, the user has to make more requests to retrieve the remainder of the result set.

Data type Integer
Default 500
Range 0 to 1024

Maximum search names:

Specifies the maximum number of names that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 8.

Higher values allow more complex requests to be processed by the UDDI node, however complex requests can significantly slow the response times of the UDDI node.

Data type Integer
Default 5
Range 1 to 64

Maximum search keys:

Specifies the maximum number of keys that can be supplied in an inquiry API request. To avoid slowing UDDI node response times, set this value no higher than 5.

This value limits the number of references that can be specified in categoryBag, identifierBag, tModelBag and discoveryURLs.

Higher values allow the UDDI node to process more complex requests, however complex requests can significantly slow the response times of the UDDI node. In exceptional cases, the UDDI node may reject complex requests with excessive numbers of keys even if the value of maxSearchKeys is not exceeded.

Data type	Integer
Default	5
Range	1 to 64

Key space requests require digital signature:

Specifies whether tModel:keyGenerator requests must be digitally signed.

Data type	Boolean (check box)
Default	False (cleared)

Use tier limits:

Specifies whether an approval manager is used to check publication tier limits. If you set this value to false, the number of UDDI entities that can be published is unlimited.

Data type	Boolean (check box)
Default	True (selected)

Use authInfo credentials if provided:

Specifies whether authInfo contents in UDDI API requests are used to validate users when WebSphere Application Server security is off. If you set this value to true, the UDDI node will use the authInfo element in the request, otherwise the default user name is used.

Data type	Boolean (check box)
Default	True (selected)

Authentication token expiry period:

Specifies the period, in minutes, after which an authentication token is invalidated and a new authentication token is required.

Set this value high enough to allow the registry to operate successfully, but be aware that high values can increase the risk of illegal use of authentication tokens.

Data type	Integer
Default	30
Range	1 to 10080 minutes (10080 minutes = 1 week)

Automatically register UDDI publishers:

Specifies whether UDDI publishers are automatically registered and assigned to the default tier. Automatically registered UDDI publishers are given default entitlements.

Data type	Boolean (check box)

Default	True (selected)

Default user name:

Specifies the user name used for publish operations when WebSphere Application Server security is disabled and **Use authInfo credentials if provided** is set to false.

Data type	String
Default	UNAUTHENTICATED

Default language code:

Specifies, for UDDI version 1 and version 2 requests, the default language code to be used for xml:lang, when not otherwise specified.

Data type	String
Default	en

Value set collection:

Use this page to view and configure the value sets that are installed in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets**.

Value sets in a UDDI node are either supported or not supported by policy. By default, new value sets are *not* supported. After you publish a value set tModel and loaded value set data, you can control whether other UDDI entities can reference this value set tModel by setting the *Supported* policy.

To enable support for one or more value sets, select the value sets by selecting the appropriate check boxes in the **Select** column. Click **Enable Support**. The Supported field for all the selected value sets is updated, with a value of true, to reflect the new status.

To disable support for a value set, which might be necessary before you remove it from the UDDI node, select the value sets in the same manner as for enabling support. Click **Disable Support**. The Supported field for all the selected value sets is updated, with a value of false, to reflect the new status.

Clicking on a value set name in the list takes you to the general properties page for that value set as described in Value set settings.

Name:

Specifies the name of the tModel that represents this value set.

tModelkey:

Specifies the key for the tModel that represents this value set.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Value set settings:

Use this page to view the attributes of a value set in a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Value Sets** > *value_set_name*.

This page shows the values of keyedReferences in the tModel that represents this value set. This page also shows the *Supported* status of the value set. All properties are read-only. To change the *Supported* status, use the Value sets collection page.

Unvalidatable:

Specifies whether this value set is unvalidatable. The value set tModel publisher sets this value, to indicate whether the value set is available for use by publish requests.

Checked:

Specifies whether this value set is checked. If you set this value to true, UDDI entities that reference this value set will be validated to ensure that their values are present in this value set.

Cached:

Specifies whether this value set is cached in this UDDI node.

Externally cacheable:

Specifies whether this value set is externally cacheable.

Externally validated:

Specifies whether this value set is externally validated.

Supported:

Specifies whether this value set is supported by policy in this UDDI node.

Last cached:

Specifies the date when this value set was last cached in the UDDI node.

Tier collection:

This page contains a list of the available tiers for the UDDI node. You can modify tiers, create new tiers and delete tiers from this page.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers**.

This page shows the available tiers for the UDDI node. Clicking a tier name will show the General Properties for the specific tier as detailed in Tier settings. To delete a Tier from the list, select the relevant name and click *Delete*. Clicking *New* will take you to the General Properties page with the same properties as described in Tier settings.

One of the tiers in the collection will be marked as the default tier, indicated by *(default)* appearing next to the tier's name. The default tier will be assigned to UDDI publishers that are registered automatically when automatic user registration is turned on. To set the default tier, select the appropriate tier in the collection and press the *Set default* button. Note that it is not possible to delete a tier if it is currently marked as the default tier, or it is currently assigned to a UDDI publisher.

Name:

Specifies the name of the tier.

Description:

Specifies the descriptive text about the tier.

UDDI Tier settings:

This topic contains details of the general properties that you can configure for a UDDI publisher tier.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Tiers** > *tier_name*.

Name:

Specifies the name of the tier.

Required	Yes
Data type	String
Default	No default
Range	1 to 255

Description:

Specifies a description of the purpose or usage of this tier.

Data type	String
Default	No default
Range	0 to 255

Maximum properties:

For each of the maximum fields described below, the data is:

Required	Yes
Data type	Integer
Default	No default
Range	0 to 2147483647

Maximum businesses:

Specifies the maximum number of businesses that UDDI publishers in this tier are allowed to publish.

Maximum services per business:

Specifies the maximum number of services that UDDI publishers in this tier are allowed to publish for each business.

Maximum bindings per service:

Specifies the maximum number of bindings that UDDI publishers in this tier are allowed to publish for each service.

Maximum tModels:

Specifies the maximum number of tModels that UDDI publishers in this tier are allowed to publish.

Maximum publisher assertions:

Specifies the maximum number of publisher assertions that UDDI publishers in this tier are allowed to add.

UDDI Publisher collection:

This page shows the users that are currently registered as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers**.

To create a UDDI publisher, click **New**. The "UDDI Publisher settings" on page 997 page, where you can enter details about the publisher, is displayed.

To register one or more existing WebSphere Application Server users as UDDI publishers, click **Create publishers**. The "Create UDDI Publishers" page, where you can select users and modify their entitlements, is displayed.

You can assign multiple publishers to a tier without editing each publisher individually, by completing the following steps:

1. Select the appropriate publishers in the collection table.
2. From the tier list at the top of the collection table, select one of the tiers that is available on the UDDI node.
3. Click **Assign tier** to update the selected publishers.

To delete publishers, select them in the collection table and then click **Delete**.

After you register the users as UDDI publishers, you can edit their entitlements as described in "UDDI Publisher settings" on page 997.

User name:

Specifies the name of the UDDI publisher.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Create UDDI Publishers:

Use this page to register one or more existing WebSphere Application Server users as UDDI publishers.

To view this administrative console page, click **UDDI** → **UDDI Nodes >** *UDDI_node_id* **> [Additional Properties] UDDI Publishers** → **Create publishers**.

To register as UDDI publishers one or more users that are known to the application server, complete the following steps:

1. Enter a string to search for the required users. To find all users, use the * character.
2. Optionally, enter a number in the limit field to restrict the number of returned results.
3. Click **Search** to display a list of users that match the string.
4. Select the users that you want to register from the **Available** list and use the arrows to move them into the **Selected** list.
5. Use the entitlements listed under **General Properties** to give the UDDI publishers permission to perform specific actions. Set the entitlements by selecting the check box next to each entitlement.

6. Select a tier for the users from the **Tier** list.
7. Click **OK** to register the users as UDDI publishers with the specified entitlements and tier.

Note: If you are using a Lightweight Directory Access Protocol (LDAP) user registry, the format of the name that is given to each UDDI Publisher is defined by the **User ID map** value in the LDAP advanced settings. To view the LDAP advanced settings, click **Security** → **Global security**, under **User account repository** select **Standalone LDAP registry** and click **Configure**, then under **Additional Properties** click **Advanced Lightweight Directory Access Protocol (LDAP) user registry settings**.

After a user has been registered as a UDDI publisher, you can edit their entitlements by clicking the user name.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string 'buyingService' is the key's key specific string (KSS).

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a domain key.

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator.

If false, UDDI publishers cannot publish keyGenerators of any kind. In this situation all the entitlement settings are disregarded, regardless of how they are set.

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel: keyGenerator with a UUID key.

Tier:

Specifies the tier to which the UDDI publisher is assigned.

UDDI Publisher settings:

Use this page to view and edit the properties of a UDDI publisher, or to create a new UDDI publisher.

You can view this administrative console page in two ways:

- If you want to view and edit the properties of an existing UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** > *user_name*
- If you want to create a new UDDI publisher click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **UDDI Publishers** → **New**

This page shows the entitlements and publication limits tier for a particular UDDI publisher.

User name:

Specifies the name of the UDDI publisher.

If you are creating a new UDDI publisher, enter the name of a user known to the application server. If you are viewing or editing the properties of an existing publisher, the user name cannot be changed.

Allowed to publish keyGenerator with derived key:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a derived key.

The tModel:keyGenerator is a request for key space. An example of a legal derived key is uddi:tempuri.com:fish:buyingService where the key is based on the derivedKey ″uddi:tempuri.com:fish″. the string 'buyingService' is the key's key specific string (KSS).

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator with domain keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a domain key.

Data type	Boolean
Default	True (selected)

Allowed to publish keyGenerator:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator.

If you set this value to false, the UDDI publisher cannot publish keyGenerators of any kind. In this situation the following permissions will be disregarded irrespective of how they are set: **Allowed to publish keyGenerator with derived key**, **Allowed to publish keyGenerator with domain keys**, **Allowed to publish with UUID key** and **Allowed to publish keyGenerator with UUID keys**.

Data type	Boolean
Default	True (selected)

Allowed to publish with UUID key:

Specifies whether the UDDI publisher has permission to publish elements with a UUID key.

Data type	Boolean
Default	False (cleared)

Allowed to publish keyGenerator with UUID keys:

Specifies whether the UDDI publisher has permission to publish tModel:keyGenerator with a UUID key.

Data type	Boolean
Default	False (cleared)

Tier:

Specifies the tier to which the UDDI publisher is assigned.

Policy groups:

This topic contains links to the detailed settings information for every policy group that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id*.

To the right of the page is a list of the Policy Groups that can be acted upon. Clicking on a specific group will open the page for the group required.

UDDI keying policy settings:

This topic contains details of the UDDI keying settings that you can configure for a UDDI registry.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Keying policies**.

Registry key generation:

Specifies whether publishers are allowed to publish key generator tModels. You can manage how publishers are allowed to publish key generator tModels by configuring the entitlements in the UDDI Publishers page.

Data type	Boolean
Default	True (selected)

Registry support of UUID keys:

Specifies whether publisher supplied uuidKeys are allowed in publish requests. You can manage how publishers are allowed to use uuidKeys by configuring the entitlements in the UDDI Publishers page.

Data type	Boolean
Default	False (cleared)

UDDI node API policy settings:

This topic contains details of the API settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** → ***UDDI_node_id*** → **API policies** .

Note: This information applies only to UDDI Version 3. You cannot change the settings for Versions 1 and 2; authentication tokens are required for publish requests, but not for inquiry requests (there are no custody transfer requests in Versions 1 or 2).

Authorization for inquiry:

Specifies whether authorization using the authInfo element is required for inquiry API requests. This setting is relevant only if the V3SOAP_Inquiry_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Inquiry_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true, an authorization token is required to complete the request. If this setting is false, an authorization token is not required; if one is supplied, it is ignored and the request is processed as if it the default user that is defined in the UDDI node settings made the request.

Typically, UDDI registries are configured to not require authorization for inquiry API requests.

Data type	Boolean
Default	False (cleared)

Authorization for publish:

Specifies whether authorization using the authInfo element is required for publish API requests. This setting is relevant only if the V3SOAP_Publish_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_Publish_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true, an authorization token is required to complete the request. If this setting is false, an authorization token is not required; if one is supplied, it is ignored and the request is processed as if it the default user that is defined in the UDDI node settings made the request.

Typically, UDDI registries are configured to require authorization for publish API requests.

Data type	Boolean
Default	True (selected)

Authorization for custody transfer:

Specifies whether authorization using the authInfo element is required for custody transfer API requests. This setting is relevant only if the V3SOAP_CustodyTransfer_User_Role is set to *Everyone* and WebSphere Application Server security is on.

If WebSphere Application Server security is off, this setting is ignored. If WebSphere Application Server security is on and the V3SOAP_CustodyTransfer_User_role is not set to *Everyone*, this setting is ignored.

If this setting is true, an authorization token is required to complete the request. If this setting is false, an authorization token is not required; if one is supplied, it is ignored and the request is processed as if it the default user that is defined in the UDDI node settings made the request.

Typically, UDDI registries are configured to require authorization for custody transfer API requests.

Data type	Boolean
Default	True (selected)

UDDI user policy settings:

This topic contains details of the user policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **User policies**.

Allow transfer of ownership:

When true, data ownership can be transferred between owners within the UDDI node.

Data type	Boolean
Default	True (selected)

UDDI data custody policy settings:

This topic contains details of the data custody settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Data custody policies**.

Transfer token expiration period:

Specifies the length of time from the issue of a transfer token, in minutes, after which that token expires.

If you set this value too high, you might expose the UDDI registry to a risk of misuse.

Data type	Integer
Default	1440
Range	1 to 2147483647 (for all intents and purposes, unlimited)

UDDI value set policy:

This topic contains details of the value set policy settings that you can configure for a UDDI registry node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* >**Value set policies**.

Enable checked value sets:

Specifies whether checked value sets are supported. If you set this value to false, publish requests of value set tModels that contain a 'checked' keyedReference will be rejected.

Data type	Boolean
Default	True (selected)

UDDI node miscellaneous:

This topic contains details of the miscellaneous settings that you can configure for a UDDI node.

To view this administrative console page, click **UDDI** → **UDDI Nodes** > *UDDI_node_id* > **Miscellaneous policies**.

Node generates discoveryURLs:

Specifies whether a UDDI node can establish a policy on whether it generates discoveryURLs.

Data type	Boolean
Default	False (cleared)

Node supports HTTP Get Service:

Specifies whether the UDDI node supports an HTTP GET service for access to the XML representations of UDDI data structures.

Data type	Boolean
Default	True (selected)

URL prefix for V3 GET servlet:

Specifies the prefix for the URL to the version 3 GET servlet that is used to retrieve the XML representation of a published entity. This property applies to UDDI version 3 API requests only.

The format of the prefix is http://hostname:port/uddiv3soap/, where uddisoap is the context root of the UDDI version 3 SOAP servlet.

When a businessEntity is published, if you have set **Node generates discovery URLs** to true, the discoveryURL value is generated based on this prefix value. Otherwise, the discoveryURL value will be empty.

The UDDI Version 3 specification recommends that you do **not** enable generation of discoveryURLs because they can affect the use of digital signatures. If you do enable generation of discoveryURLs, do not change the URL prefix afterward, otherwise discoveryURLs that were generated using the earlier URL prefix no longer work.

Data type	URL
Default	http://localhost:9080/uddiv3soap/

Backing up and restoring the UDDI registry database

If you want to protect the data in your UDDI registry database, you can back up and restore the database using the facilities of the database product that your UDDI node is on.

- To backup a Apache Derby UDDI registry database, first ensure that the UDDI application is stopped (and hence, not accessing the Apache Derby database), and ensure that no other application is using the Apache Derby UDDI30 database. Make a copy of the UDDI30 directory using the file system that the directory resides upon. To restore the database, replace the UDDI30 file structure with the back up. Note that any updates made since the back up was taken will be lost.
- To backup a non-Apache Derby UDDI registry database, use the appropriate backup and restore tools for the database. To use these tools, refer to the documentation for the database product.

Enabling Web services through the service integration bus

Web services can use the service integration bus to provide a single point of control, access, and validation of Web service requests and allow control of Web services that are available to different groups of Web service users.

About this task

With bus-enabled Web services you can achieve the following goals:

- Create an *inbound service*: Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Create an *outbound service*: Take an externally-hosted Web service, and make it available internally at a bus destination.

Bus-enabled Web services provide a choice of quality of service and message distribution options, along with intelligence in the form of mediations that allow for the rerouting of messages.

To enable Web services through service integration technologies, complete the following steps:

1. Optional: Learn about bus-enabled Web services. Explore the concepts that underly service integration bus-enabled Web services.
2. Plan your bus-enabled Web services installation. Determine the bus-enabled Web services roles that each server is to perform.
3. Ensure that every server that is to play a bus-enabled Web services role is a member of a service integration bus. For more information, see “Configuring the members of a bus” on page 1145.

4. For every server that is to play a bus-enabled Web services role, install and configure a Service Data Objects (SDO) repository on the server.

   Note: For WebSphere Application Server Version 6.0, you also had to manually install a selection of the following applications:
   - The service integration technologies resource adapter (used to invoke Web services at outbound ports).
   - The bus-enabled Web services application.
   - One or more endpoint listener applications.

   For later versions of WebSphere Application Server, these applications are installed automatically as and when needed. For example, the endpoint listener application is installed automatically when you configure an endpoint listener.
5. Create a new endpoint listener configuration for each endpoint listener application that you plan to use to receive inbound service requests.
6. Optional: Create an inbound service. An inbound service is a Web interface to a service that is provided internally (that is, a service provided by your own organization and hosted in a location that is directly available through a service integration bus destination). To configure a locally-hosted service as an inbound service, you associate it with a service destination, and with one or more endpoint listeners through which service requests and responses are passed to the service. You can also choose to have the local service made available through one or more UDDI registries.
7. Optional: Create an outbound service. An outbound service is a Web service that is hosted externally, and is made available through a service integration bus. To make an externally-hosted service available through a bus, you first associate it with a service destination, then you configure one or more port destinations (one for each type of binding, for example SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service. You get the port definitions from the WSDL, but you can choose which ones you want to create.
8. Optional: Apply additional security to your bus-enabled Web services. By default, the bus-enabled Web services configuration works when WebSphere Application Server security is enabled and your service integration buses are secured. However this level of security does not impose any security restrictions on the users of your bus-enabled Web services configuration. To control how your bus-enabled Web services configuration is used by each group of your colleagues or customers, use the bus-enabled Web services additional security features to enable working with password-protected components and servers, with WS-Security and with HTTPS.

What to do next

For further information on specific aspects of bus-enabled Web services, see the following topics:
- Learning about bus-enabled Web services
- “Installing and configuring the SDO repository”
- “Configuring Web services for a service integration bus” on page 1008
- “Administering the bus-enabled Web services core resources” on page 1019
- Securing bus-enabled Web services
- “Passing SOAP messages with attachments through the service integration bus” on page 1052
- SIBWebServices command group for the AdminTask object
- Bus-enabled Web services troubleshooting tips
- Tuning bus-enabled Web services

Installing and configuring the SDO repository

Service Data Objects (SDO) is an open standard for enabling applications to handle data from different data sources in a uniform way, as DataGraphs. Service integration bus-enabled Web services use an SDO

repository for storing and serving WSDL definitions. Use this task to create and configure your preferred database to store SDO data, and to install and configure an SDO repository on each server that you plan to use for bus-enabled Web services.

Before you begin

Determine the servers on which to install and configure an SDO repository as described in Planning your bus-enabled Web services installation, then add each server as a member of a bus as described in "Configuring the members of a bus" on page 1145.

An SDO repository can work with most database products. For specific information about choosing and configuring your preferred database, consult your database administrator or database product documentation, as well as reading the notes on database usage given in this topic.

About this task

To install and configure an SDO repository, you complete the following broad steps:

- Install your preferred database product.
- Create a JDBC provider and a data source for your database.
- Run the `installSdoRepository.jacl` script one or more times, to install the SDO application on each server and to set the database type that the SDO repository is to use.

For detailed information about how to do this, first read the following notes on database usage and on the `installSdoRepository.jacl` script, then complete the steps for one of these configurations:

- "Configure the SDO repository for a single server, using the embedded Derby database" on page 1005.
- "Configure the SDO repository for a single server, using a database other than embedded Derby" on page 1005.

Note:

- For a single server configuration you can use either your preferred database or the embedded Apache Derby database that is supplied with WebSphere Application Server.
- The SDO repository dictates the schema and table names it uses, so different repositories must use different databases to ensure they do not access the same data.
- Create the database for your preferred database supplier using the `Table.ddl` file from the relevant *app_server_root*`/util/SdoRepository/`*database_type* directory. The `Table.ddl` file describes the database table that is needed by the SDO repository.
- The `-editBackendId` flag on the `installSdoRepository.jacl` script determines the database type that the repository is to use. The back end ID determines what database-specific rules the application follows when talking to the database. See the associated note on the `installSdoRepository.jacl` script.
- Some databases require that a user ID be created and granted permissions to access the SDO repository database. Create a user ID for user name `SDOREP` before you create the tables for Oracle, Sybase, and SQL Server databases. Because of the way these databases handle user names and table names, the user name must be `SDOREP` to enable the SDO repository to access its table with the fully qualified name `SDOREP.BYTESTORE`. Make sure you grant permission for the `SDOREP` user to read from, and write to, the database.
- If you use an Informix® database, do not disable logging.
- The SDO repository does not require *XA* support. In most cases you can use either an XA or a non-XA data source; however if your database is Oracle 8 or 9 then you must use the Oracle JDBC driver (non-XA) for the SDO repository data source.
- You might also choose to perform other steps such as creating an index of the primary key to improve database performance. Do not change the schema, table and column names.

Note:

- Use the wsadmin scripting client to run the script.
- Run the script from within QShell.
- The script is provided in the *app_server_root*/bin directory, where *app_server_root* is the root directory for the installation of WebSphere Application Server. If you choose to run the wsadmin scripting client from another directory, specify the full path to the script on the command option. For example to work with a profile other than the default profile, change to the *app_server_root*/profiles/*profile_name*/bin directory then specify the following path to the script:

  ```
  wsadmin -f app_server_root/bin/installSdoRepository.jacl
  ```
- The -editBackendId flag on the installSdoRepository.jacl script determines the database type that the repository is to use. The back end ID determines what database-specific rules the application follows when talking to the database. To see the full list of available back end ID values, use the -listBackendIds flag:

  ```
  wsadmin -f installSdoRepository.jacl -listBackendIds
  ```
- If the data source already exists, or there has been a previous broken or partial installation of the SDO repository application, the installSdoRepository.jacl script fails to complete and configuration changes are not saved. In these cases, run the SDO repository uninstall script, fix the problem, then rerun the installSdoRepository.jacl script.

Configure the SDO repository for a single server, using the embedded Derby database

About this task

If you are creating a single server configuration and you want to use embedded Derby, you run the installSdoRepository.jacl script with the -createDb switch. This action creates the Derby database and installs the SDO repository.

To configure the SDO repository for a single server using the embedded Derby database, complete the following steps:

1. Open a command prompt, then change to the *app_server_root*/bin directory.
2. Enter the following command:

```
wsadmin -f installSdoRepository.jacl -createDb
```

Note: The -createDb flag tells the command to create a default Derby database. If you omit this flag, the command still installs an SDO repository that is configured to use Derby, but the command does not also create the database.

Configure the SDO repository for a single server, using a database other than embedded Derby

About this task

If you are creating a single server configuration that uses a database other than embedded Derby, you install your preferred database product, then create a JDBC provider and a data source, then run the installSdoRepository.jacl script twice:

1. One time to install the SDO application on the application server.
2. One time to set the database type that the SDO repository is to use.

To configure the SDO repository for a single server using a database other than embedded Derby, complete the following steps:

1. Create the database for your preferred database supplier using the Table.ddl file from the relevant *app_server_root*/util/SdoRepository/*database_type* directory.

For an illustration of the process for creating tables in DB2, see Recreating database tables from the exported table data definition language. For more information, see Deploying data access applications.

2. Create a J2C authentication alias.

   This is for use with the data source that you create in the next step. Check that the authentication alias matches the login details for your database instance, otherwise a connection will not be made.

3. Create and configure a JDBC provider and data source.

   Set the following data source properties:

   - Set the **authentication** property to use the authentication alias you created in the previous step.
   - Select the **Use this Data Source in container managed persistence (CMP)** check box.
   - Set the **Name** property to a name of your own choosing. For example, `SDO Repository DataSource`.
   - Set the **JNDI name** property to the following exact value: `jdbc/com.ibm.ws.sdo.config/SdoRepository`.
   - Set any other properties that are required settings for your chosen database.

4. Optional: Test the data source connection:

   Note: This option does not work in all configurations. The availability of this option depends on the scope at which the data source is defined, and the scope of any WebSphere Application Server variables that are used in the JDBC provider and data source configurations. For more information about testing connections to data sources, see Test connection service.

   a. In the administrative console, navigate to **Resources** → **JDBC** → **Data sources**.

   b. Select the SDO repository data source.

   c. Click **Test connection**.

5. Configure the SDO repository:

   a. Open a command prompt, then change to the *app_server_root*`/bin` directory.

   b. Install the SDO repository application on the server:

```
wsadmin -f installSdoRepository.jacl
```

   c. Set the database type that the SDO repository is to use:

```
wsadmin -f installSdoRepository.jacl -editBackendId database_type
```

   for example:

```
wsadmin -f installSdoRepository.jacl -editBackendId DB2UDB_V82
```

The SDO repository uninstall script

Use this script to uninstall a Service Data Objects (SDO) repository that was previously installed, or failed to install correctly.

You install the SDO repository application on every server that you plan to use for one or more of the service integration bus-enabled Web services roles as described in “Installing and configuring the SDO repository” on page 1003.

If the data source already exists, or there has been a previous broken or partial installation of the SDO repository application, the `installSdoRepository.jacl` script fails to complete and configuration changes are not saved. In these cases, you need to run the `uninstallSdoRepository.jacl` script. This script continues when it finds unexpected results, so it can clean up a broken or partial installation.

Note: Run the script from within QShell.

The script is provided in the *app_server_root*`/bin` directory, where *app_server_root* is the root directory for the installation of WebSphere Application Server. If you choose to run the wsadmin scripting client from

another directory, specify the full path to the script on the command option. For example to work with a profile other than the default profile, change to the *app_server_root*/profiles/*profile_name*/bin directory then specify the following path to the script:

```
wsadmin -f app_server_root/bin/uninstallSdoRepository.jacl
```

The SDO repository script install and uninstall pairs

The following are the install and uninstall command pairs, where each uninstall command undoes the action of the related install command. If you attempt to uninstall with a different set of arguments to those previously used with the installSdoRepository.jacl script, you might find that the uninstall does not remove everything or that it displays warnings when it tries to remove non-existent settings.

For configuration of the SDO repository on a server, the -createDb flag tells the install command to create a default (Apache Derby) database and configure it for use with this application server. The -removeDb flag tells the uninstall command to remove the database configuration from the application server, but not to delete the Apache Derby database:

```
wsadmin -f installSdoRepository.jacl -createDb
wsadmin -f uninstallSdoRepository.jacl -removeDb
```

Note:

- If you did not use -createDb on the installer, because you had already configured an Apache Derby database for some other purpose, then you should not use the -removeDb flag on the uninstaller.
- To avoid deleting data that you might want to keep, the -removeDb flag does not delete the Apache Derby database. If you are certain that you want to delete the database, you can do so manually. An Apache Derby database is a directory on the file system. The one created by the installer with the -createDb flag is in the *profile_root*/databases/SdoRepDb directory, where *profile_root* is the directory in which profile-specific information is stored. If you do not delete the database and you try to install again with the -createDb flag, the install fails stating that the SdoRepDb directory already exists.

For installation or removal of the SDO repository application from a server:

```
wsadmin -f installSdoRepository.jacl
wsadmin -f uninstallSdoRepository.jacl
```

Bus-enabled Web services installation files and locations

When you install WebSphere Application Server, the files that are required for running service integration bus-enabled Web services are copied into your file system under *app_server_root*, where *app_server_root* is the root directory for the installation of WebSphere Application Server.

The following table lists the main bus-enabled Web services files that you might need to access, and the locations into which they are placed.

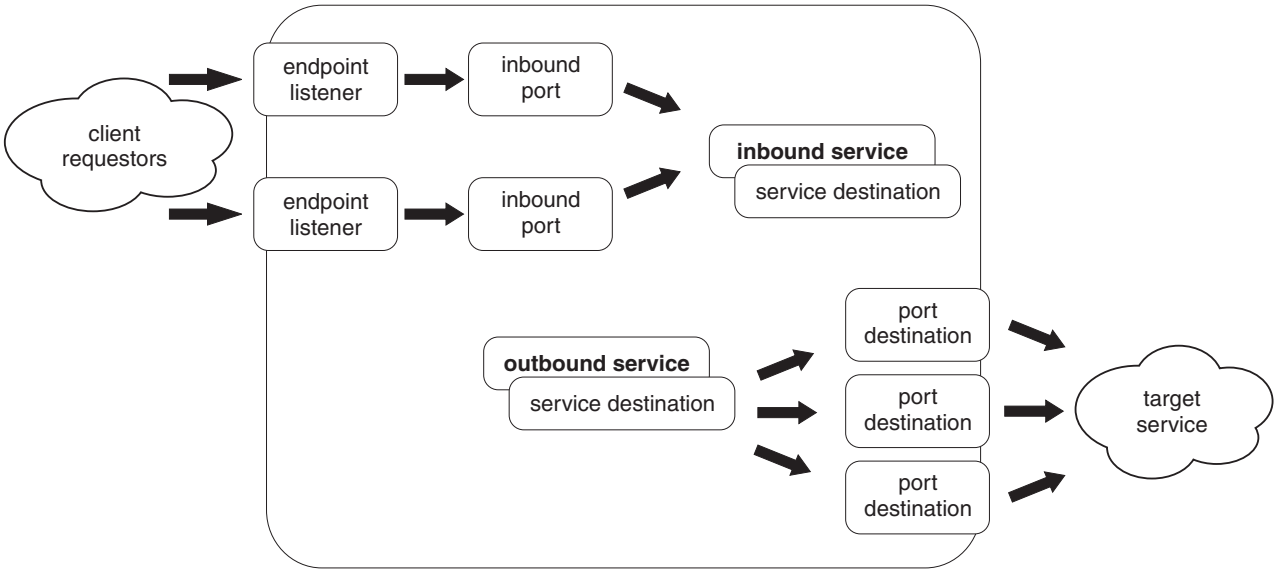
File name	Purpose	Location
installSdoRepository.jacl	The script used to configure the SDO repository. Bus-enabled Web services use an SDO repository for storing and serving their WSDL definitions.	*app_server_root*/bin
uninstallSdoRepository.jacl	The script used to uninstall the SDO repository.	*app_server_root*/bin
SDO repository resources	The resource files used to configure the SDO repository to work with a range of different databases.	*app_server_root*/util/SdoRepository

File name	Purpose	Location
sibwsauthbean.ear	The application for Password-protecting a Web service operation.	*app_server_root*/installableApps
sibwsAuthGen	The command file used to generate authorization beans for password-protecting a Web service operation.	*app_server_root*/util
soaphttpchannel1.ear	The SOAP over HTTP endpoint listener 1 application. This application is installed automatically when you create an associated endpoint listener configuration.	*app_server_root*/installableApps
soaphttpchannel2.ear	The SOAP over HTTP endpoint listener 2 application. This application is installed automatically when you create an associated endpoint listener configuration.	*app_server_root*/installableApps
soapjmschannel1.ear	The synchronous SOAP over Java Messaging Service (JMS) endpoint listener 1 application. This application is installed automatically when you create an associated endpoint listener configuration.	*app_server_root*/installableApps
soapjmschannel2.ear	The synchronous SOAP over JMS endpoint listener 2 application. This application is installed automatically when you create an associated endpoint listener configuration.	*app_server_root*/installableApps

Configuring Web services for a service integration bus

Take an internally-hosted service that is available at a bus destination, and make it available as a Web service; Take an externally-hosted Web service, and make it available internally at a bus destination; Use the Web services gateway to map an existing service - either an inbound or an outbound service - to a new Web service that appears to be provided by the gateway.

About this task



Through service integration bus-enabled Web services you can achieve the following goals:

- Create an *inbound service*: Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Create an *outbound service*: Take an externally-hosted Web service, and make it available internally at a bus destination.

Making an internally-hosted service available as a Web service

Create an inbound service. An inbound service is a Web interface to a service that is provided internally (that is, a service provided by your own organization and hosted in a location that is directly available through a service integration bus destination). To configure a locally-hosted service as an inbound service, you associate it with a service destination, and with one or more endpoint listeners through which service requests and responses are passed to the service. You can also choose to have the local service made available through one or more UDDI registries.

Before you begin

This topic assumes that:

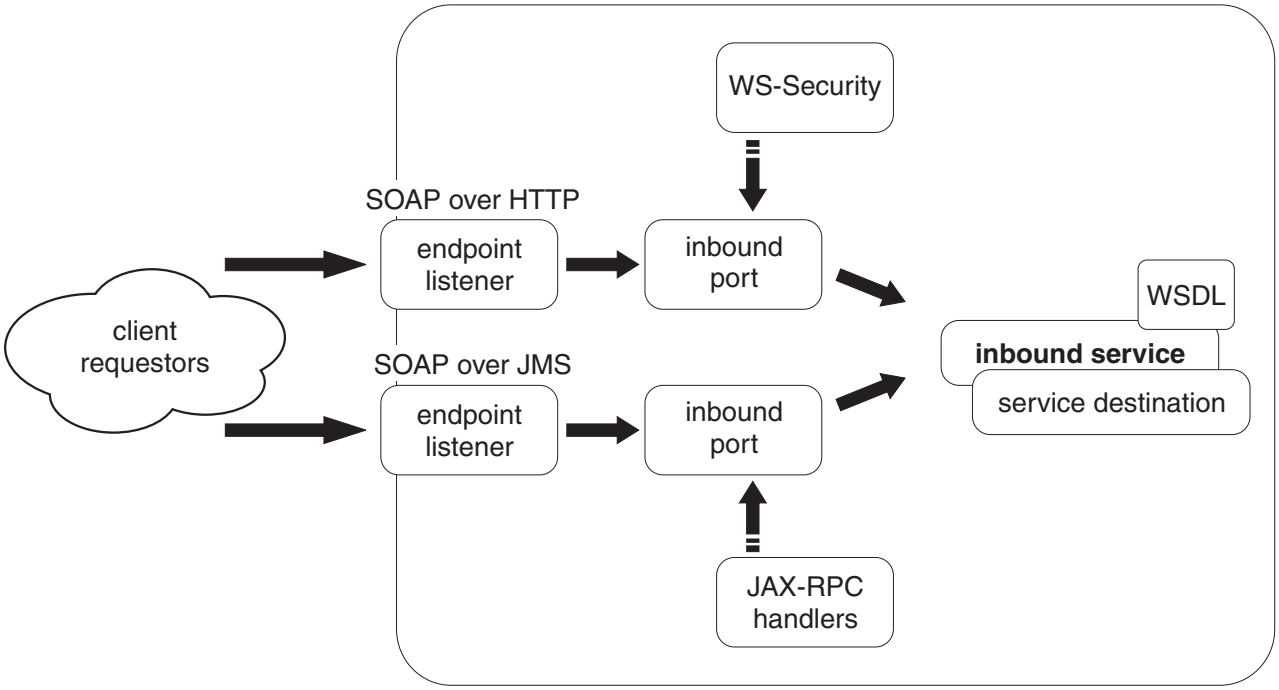
- You have created and installed a Service Data Objects (SDO) repository (used for storing and serving WSDL definitions) on every server that is to play a service integration bus-enabled Web services role.
- You have created a new endpoint listener configuration for each endpoint listener that you plan to use to receive inbound service requests.
- You already have an internally-hosted service that you want to configure as an inbound service, and you have made the service available at a service integration bus destination.
- You have created references to any UDDI registries in which you want to register this service.

You must also create a template WSDL file that describes the service, and make the WSDL available at a URL or through a UDDI registry. For information on how to create a WSDL file, see Developing a WSDL file.

You can create an inbound service by using the administrative console as described in this task, or by using the createSIBWSInboundService command.

Note: If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you cannot use the administrative console for this task and you must create your new inbound service by using the wsadmin tool. For more information see the corresponding troubleshooting tip.

About this task



service integration technologies

Web service requests and responses to an inbound service can be sent across any binding (for example SOAP over HTTP or SOAP over JMS) that is available to the bus. Each available binding type is represented by an inbound port, and each inbound port is associated with a binding-specific endpoint listener.

You can control and monitor access to your inbound services in the following ways:

- You can control which groups of users can access a particular inbound Web service by making the service available only through specific endpoint listeners.
- You can associate JAX-RPC handler lists with ports, so that the handlers can monitor activity at the port, and take appropriate action depending upon the sender and content of each message that passes through the port.
- You can set the level of security to be applied to messages (the WS-Security configuration and bindings). The security level can be set independently for request and response messages.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Inbound Services**. The inbound services collection form is displayed.
3. Click **New**. The New inbound service wizard is displayed.
4. Use the wizard to create the new inbound service configuration by completing the following steps. For more information about the properties that you set with the wizard, see Inbound services settings.
   a. Select the service destination and template WSDL location.

      Note: The template WSDL is the service-specific WSDL file that you have created to describe this inbound service.

b. Select the service from the template WSDL.

Note:

- This option is needed in case there is more than one service in the template WSDL. The field is filled in for you by default. If there is only one service in the WSDL, accept the default.

c. Specify the name of the inbound service and select the endpoint listeners.

Note:

- You need not supply a name for the inbound service. If you choose not to supply a name, a default name is created. The default name is derived from the service destination name, with characters that are not valid for names filtered out.
- An inbound port is automatically created for each endpoint listener that you select. Each inbound port is created without a template port, JAX-RPC handler list or security settings and is given a default name that relates to the endpoint listener selected. For an overview of the relationship between endpoint listeners and inbound ports, see Endpoint listeners and inbound ports: Entry points to the service integration bus.

d. Define any UDDI publication properties.

Note: The wizard allows you to specify the UDDI publication properties that are used to publish this inbound service to an initial UDDI registry. After you create an inbound service through the wizard, you can use the modify an existing inbound service configuration option to publish the service to more UDDI registries. For information about the UDDI publication properties, see UDDI Publication settings and UDDI registries: Web service directories that can be referenced by bus-enabled Web services.

5. Click **Finish**.

Results

If the processing completes successfully, the list of inbound services for this service integration bus is updated to include the new inbound service. Otherwise, an error message is displayed.

What to do next

If you want to secure your new inbound service, or apply any JAX-RPC handler lists to the ports for the service, or publish the service to more UDDI registries, use the administrative console to modify your inbound service configuration.

Modifying an existing inbound service configuration:

Modify the configuration details for an existing inbound service. For example: secure the service; apply JAX-RPC handler lists to the ports for the service; publish the service to more than one UDDI registry.

About this task

An inbound service is a Web interface to a service that is provided internally (that is, a service provided by your own organization and hosted in a location that is directly available through a service integration bus destination).

When you first create an inbound service, you connect the service to one or more endpoint listeners and (optionally) specify the UDDI publication properties that are used to publish the inbound service to an initial UDDI registry. An inbound port is automatically created for each endpoint listener that you select, but each inbound port is created without a template port, JAX-RPC handler list or security settings. You need to modify your inbound service configuration if you want to control and monitor access to your inbound services in any of the following ways:

- Associate JAX-RPC handler lists with ports, so that the handlers can monitor activity at the port, and take appropriate action depending upon the sender and content of each message that passes through the port.
- Password-protect a Web service operation.
- Set the level of security to be applied to messages (the WS-Security configuration and bindings). The security level can be set independently for request and response messages.
- Publish the service to more than one UDDI registry.

To list the existing inbound services, and to view and modify their configuration details, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Inbound Services**. A list of all the inbound services is displayed in an inbound services collection form.
3. Click the name of an inbound service in the list. The current settings for this inbound service are displayed.
4. Optional: Click **Reload template WSDL** to reload the template WSDL file for this inbound service.

   Note:

   - When you create a new inbound service, a copy of the template WSDL file for the service is loaded into a locally-maintained repository. If you change the template WSDL file, you must update the local copy.
   - When you click **Reload template WSDL**, you launch the command that is described in Refreshing the inbound service WSDL file by using the wsadmin tool. For the command to complete successfully, the conditions must be met that are described in that topic.
   - If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you cannot use the **Reload template WSDL** option and you must launch the refresh WSDL command by using the wsadmin tool. For more information, see the corresponding troubleshooting tip.
5. Modify the general properties. For information about each of these properties, see Inbound services settings.

   Note:

   - When you change an inbound service name, the system looks up all objects that refer to it and updates the name.
   - The template WSDL is the service-specific WSDL file that you create to describe this inbound service. For information on how to create a WSDL file, see Developing a WSDL file.
   - Although logically the template WSDL name and namespace are only required if there is more than one service in the WSDL, the fields that you use to set them are coded within the administrative console as compulsory fields. They are filled in for you by default, so if they are not logically required for your service you should leave the default values. If you remove the value from either field, the administrative console treats the empty field as an error.
   - If you select the option to **Enable operation-level security** then you must also complete, for this inbound service, the steps described in Password-protecting a Web service operation.
6. Modify the additional properties.
   a. Modify the inbound ports that are associated with this inbound service.

      An inbound port describes the Web service enablement of a service destination on a specific endpoint listener, with associated configuration. Each inbound port is associated with an endpoint listener, and you can control which groups of users can access a particular inbound service by making the service available only through specific endpoint listeners. For more information, see Endpoint listeners and inbound ports: Entry points to the service integration bus.

You can use a JAX-RPC handler list to monitor activity at the port, and take appropriate action (for example logging, or re-routing) depending upon the sender and content of each message that passes through the port. For more information, see Bus-enabled Web services and JAX-RPC handlers.

You can use WS-Security to set the levels of security to be applied to messages. The security level can be set independently for request and response messages. For more information, see Service integration technologies and WS-Security.

See also Inbound ports settings.

b. Modify the UDDI publication properties that are used to publish this inbound service to one or more UDDI registries. For information about the UDDI publication properties, see UDDI Publication settings and UDDI registries: Web service directories that can be referenced by bus-enabled Web services.

c. Modify the custom properties, if any, that you have set for this inbound service. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value.

d. Use the publish WSDL files property to export the template WSDL for this inbound service to a ZIP file.

As a technology preview, the exported ZIP file includes a version of the WSDL file that has no ports (bindings) defined. This non-bound WSDL is intended for use by your colleagues preparing to deploy an inbound service. It gives you a convenient way of sharing information on the planned deployment details for the service among your team. When you finally deploy the inbound service, the associated WSDL must be complete (that is, it must include the binding information).

The non-bound WSDL file is always published in the exported ZIP file for the inbound service, along with the bound WSDL file if the inbound service has any ports defined. The ZIP file, named *inbound_service_name*.zip, therefore always contains the following files:

- *bus_name*.*inbound_service_name*NonBound.wsdl (this file contains the non-bound service, port and binding for the inbound service).
- *bus_name*.*inbound_service_name*PortTypes.wsdl (this file contains the port type definition for the inbound service).

If the inbound service has one or more ports, then the ZIP file additionally contains the following files:

- *bus_name*.*inbound_service_name*Service.wsdl (this file contains the service and port elements for the inbound service).
- *bus_name*.*inbound_service_name*Bindings.wsdl (this file contains the binding elements that correspond to the ports for the inbound service).

If there is an error generating the WSDL then an error page is returned.

7. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of inbound services for this service integration bus is redisplayed. Otherwise, an error message is displayed.

Deleting inbound services configurations:
Before you begin

Decide on which method you want to use to configure these resources. You can delete an inbound service by using the administrative console as described in this task, or by using the deleteSIBWSInboundService command.

About this task

To delete one or more inbound services by using the administrative console, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Inbound Services**. A list of all the inbound services is displayed in an inbound services collection form.
3. Select the check box for every inbound service that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of inbound services for this service integration bus is updated. Otherwise, an error message is displayed.

Making an externally-hosted Web service available internally

Create an outbound service. An outbound service provides access, through one or more outbound ports, to a Web service that is hosted externally. An outbound service can be used by any of your internal systems that can access the service integration bus on which it is hosted. To make an externally-hosted service available through a bus, you first associate it with a service destination, then you configure one or more port destinations (one for each type of binding, for example SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service. You get the port definitions from the WSDL, but you can choose which ones you want to create.

Before you begin

This topic assumes that you have created and installed a Service Data Objects (SDO) repository (used for storing and serving WSDL definitions) on every server that is to play a service integration bus Web services role.

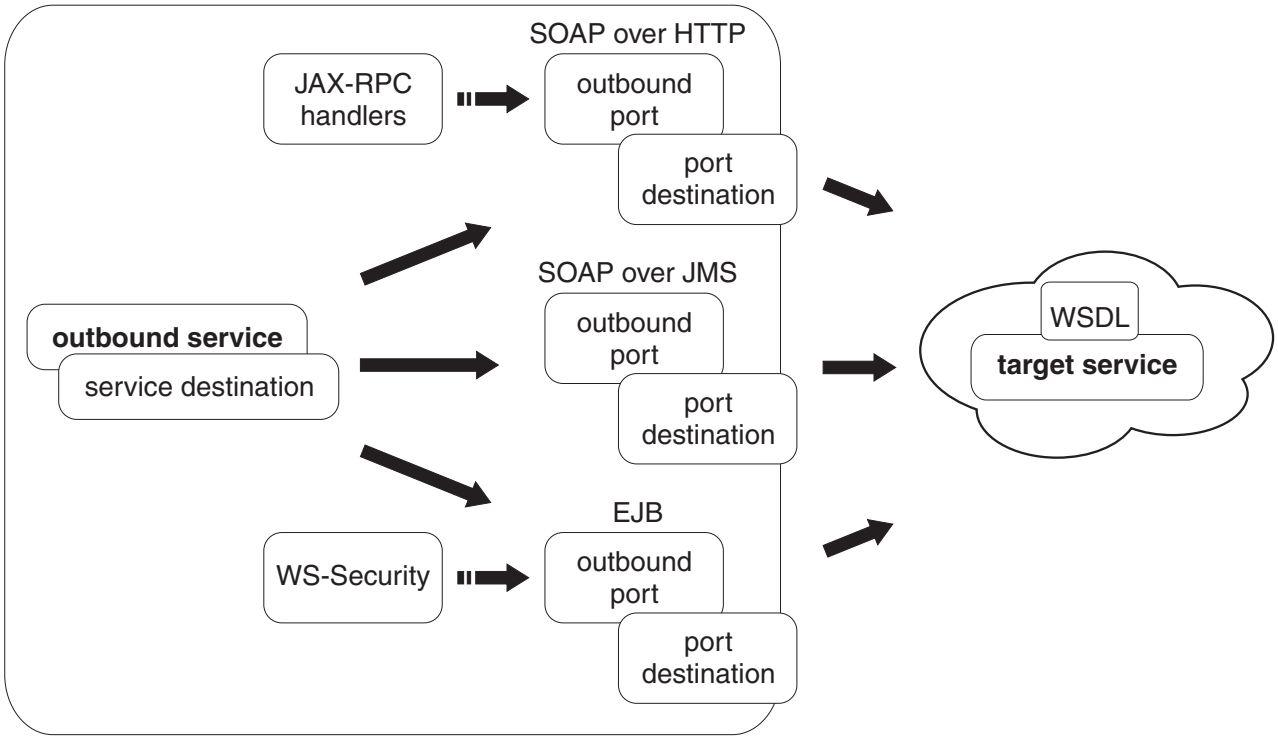
To create an outbound service, you need to know the location of the externally-published WSDL file that describes the service. This WSDL file is either available at a Web address or through a UDDI registry.

If the WSDL file for your outbound service is stored in a UDDI registry, you associate the outbound service with a UDDI reference to the registry. You choose the UDDI reference from a selection list, so you must configure the UDDI reference before you configure a new outbound service that uses it.

Decide on which method you want to use to configure these resources. You can create an outbound service by using the administrative console as described in this task, or by using the createSIBWSOutboundService command.

Note: If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you cannot use the administrative console for this task and you must create your new outbound service by using the wsadmin tool. For more information see the corresponding troubleshooting tip.

About this task



service integration technologies

Requests and responses to an outbound service are sent across any transport binding (for example SOAP over HTTP, SOAP over JMS, EJB binding) that is available to both the target service and the service integration bus. Each available binding type is represented by an outbound port configured at a port destination. For more information, see Outbound ports and port destinations.

You can control and monitor access to the target service in the following ways:

- You can associate JAX-RPC handler lists with ports, so that the handlers can monitor activity at the port, and take appropriate action depending upon the sender and content of each message that passes through the port.
- You can set the level of security to be applied to messages (the WS-Security binding). The security level can be set independently for request and response messages.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services**. The outbound services collection form is displayed.
3. Click **New**. The New outbound service wizard is displayed.
4. Use the wizard to create the new outbound service configuration by completing the following steps. For more information about the properties that you set with the wizard, see Outbound services settings and Outbound ports settings.
   a. Locate the target service WSDL.
   b. Select the service from the WSDL.

      Note:

      - This option is needed in case there is more than one service in the WSDL. The field is filled in for you by default. If there is only one service in the WSDL, accept the default.
      - There needs to be at least one port defined in the service you select.

c. Select the ports that are to be enabled for this service.

   Note: Select at least one port.

d. Name the outbound service, the service destination and all of the port destinations.

   Note:

   - Default names are generated, but you can rename them. The default names are unique within the current service integration bus. Any replacement names that you choose must be similarly unique. If you enter a name that is not unique, an error message is displayed.
   - If you have created a port selection mediation and deployed it to the service integration bus, then it is available for selection in the list of mediations. If you do not want to use a port selection mediation with this outbound service, select `none` from the selection list. This list contains all mediations, including port selection mediations, that are currently deployed to this service integration bus.
   - The list of available ports is a subset of the ports that are described in the WSDL file. You chose this subset in the previous step. If you selected more than one port in the previous step, you should also set the default port to be used unless otherwise specified by a port selection mediation.

e. Assign each port destination and (optionally) the port selection mediation to a bus member.

   Note:

   - Bus members are application servers or clusters that are added to this bus.
   - The option to assign a port selection mediation to a bus member is only displayed if you selected a mediation in the previous step.

5. Click **Finish**.

Results

If the processing completes successfully, the list of outbound services for this service integration bus is updated to include the new outbound service. Otherwise, an error message is displayed.

What to do next

Because the service is hosted externally, you might also need to enable proxy server authentication for each port to get permission to access the Internet.

If you want to secure your new outbound service, or apply any JAX-RPC handler lists to the ports, or enable proxy server authentication for any of the ports, use the administrative console to modify your outbound service configuration.

Modifying an existing outbound service configuration:

Modify the configuration details for an outbound service. For example: secure the service; apply JAX-RPC handler lists to the ports for the service; publish the service to more than one UDDI registry.

About this task

An outbound service provides access, through one or more outbound ports, to a Web service that is hosted externally. An outbound service can be used by any of your internal systems that can access the service integration bus on which it is hosted.

When you first create an outbound service you select the ports that are to be enabled for the service, but you do not associate the ports with JAX-RPC handler lists or security settings. You need to modify your outbound service configuration if you want to control and monitor access to the target service in any of the following ways:

- Associate JAX-RPC handler lists with ports, so that the handlers can monitor activity at the port, and take appropriate action depending upon the sender and content of each message that passes through the port.
- Password-protect a Web service operation.
- Set the level of security to be applied to messages (the WS-Security binding). The security level can be set independently for request and response messages.
- Enable proxy server authentication for any of the ports.

To list the existing outbound services, and to view and modify their configuration details, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services**. A list of outbound services is displayed in an outbound services collection form.
3. Click the name of an outbound service in the list. The current settings for this outbound service are displayed.
4. Optional: Click **Reload WSDL** to reload the external WSDL file for this outbound service.

   Note:

   - When you create a new outbound service, a copy of the external WSDL file for the service is loaded into a locally-maintained repository. If the external service provider changes the WSDL file, you must update the local copy.
   - When you click **Reload WSDL**, you launch the command that is described in Refreshing the outbound service WSDL file by using the wsadmin tool. For the command to complete successfully, the conditions must be met that are described in that topic.
   - If the bus needs to pass messages through an authenticating proxy server to retrieve WSDL documents, then you cannot use the **Reload WSDL** option and you must launch the refresh WSDL command by using the wsadmin tool. For more information see the corresponding troubleshooting tip.
5. Modify the general properties. For information about each of these properties, see Outbound services settings.

   Note:

   - When you change an outbound service name, the system looks up all objects that refer to it and updates the name. Any replacement name that you choose must be unique within the current service integration bus. If you enter a name that is not unique, an error message is displayed.
   - You cannot change the **Service destination name**. However, if you click **View** alongside the name, you can view and modify the configuration information for the service destination.
   - If you change the WSDL location information (that is the fields **WSDL location type**, **WSDL location** and **WSDL UDDI Registry**), then click **Apply**, the outbound service WSDL file is reloaded. Therefore you should click **Apply** after you make any changes to the WSDL location information and before you change any of the WSDL-derived fields (for example WSDL service name, and list of available ports).
   - Although logically the WSDL service name and namespace are only required if there is more than one service in the WSDL, the fields that you use to set them are coded within the administrative console as compulsory fields. They are filled in for you by default, so if they are not logically required for your service you should leave the default values. If you remove the value from either field, the administrative console treats the empty field as an error.

- The list of available ports from which you choose the **Default port name** is a subset of the ports that are described in the WSDL file. You chose this subset when you created or last modified this outbound service. To add or remove available ports, use the additional properties option **Outbound Ports**.
- If you have created a port selection mediation and deployed it to the service integration bus, then it is available for selection in the list of mediations. If you do not want to use a port selection mediation with this outbound service, select `none` from the selection list. This list contains all mediations, including port selection mediations, that are currently deployed to this service integration bus.
- Bus members are application servers or clusters that are added to this bus. The **Bus member** property defines the bus member to which the port selection mediation is assigned. If you change the **Port selection mediation** property value to `(none)`, you should also change the **Bus member** property value to `(none)`. If you want to use a port selection mediation, assign it to a bus member. If you do not do this, the administrative console displays an error message.
- If you select the option to **Enable operation-level security** then you must also complete, for this outbound service, the steps described in Password-protecting a Web service operation.
-

6. Modify the additional properties.
   a. Modify the ports that are associated with this outbound service. For information about the properties of outbound service ports, see Outbound ports settings.

      Note:

      - Requests and responses to an outbound service can be sent across any binding (for example SOAP over HTTP or SOAP over JMS) that is available to both the service integration bus and the external Web service. Each available binding is represented by a port.
      - You can use a JAX-RPC handler list to monitor activity at the port, and take appropriate action (for example logging or re-routing) depending upon the sender and content of each message that passes through the port. If the external Web service requires HTTP basic authentication, you can use a JAX-RPC handler list to provide an HTTP basic authentication header as described in Invoking a password-protected outbound service.
      - You can use WS-Security to set the levels of security to be applied to messages. The security level can be set independently for request and response messages. For more information, see Service integration technologies and WS-Security.
      - You can set the levels of security to be applied to messages. The security level can be set independently for request and response messages.
      - The service integration technologies require access to the Internet to invoke an outbound service or to retrieve a target service WSDL file. If you use a proxy server in support of Internet routing, and if your proxy server requires authentication before it grants access to the Internet, then you must enable proxy server authentication.

   b. Modify the custom properties, if any, that you have set for this outbound service. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value.
7. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of outbound services for this service integration bus is redisplayed. Otherwise, an error message is displayed.

Deleting outbound service configurations:

Before you begin

Decide on which method you want to use to configure these resources. You can delete an outbound service by using the administrative console as described in this task, or by using the deleteSIBWSOutboundService command.

About this task

To delete one or more outbound services by using the administrative console, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Services] Outbound Services**. A list of outbound services is displayed in an outbound services collection form.
3. Select the check box for every outbound service that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of outbound services for this service integration bus is updated. Otherwise, an error message is displayed.

Administering the bus-enabled Web services core resources

Use the administrative console to configure the service integration bus-enabled Web services core resources: endpoint listeners; JAX-RPC handler lists; WS-Security bindings and configurations; references to UDDI registries.

About this task

When you configure a bus-enabled Web service you associate it with one or more of the following core resources:

- The endpoint listeners on which you want the service to be available.
- Any JAX-RPC handler lists that apply to the service.
- Any WS-Security bindings and configurations that apply to the service.
- Any references to UDDI registries in which entries for the service are created.

You choose each of these resources from a list of resources that you have previously configured.

Use the administrative console to administer the bus-enabled Web services core resources as described in the following topics:

1. “Creating a new endpoint listener configuration.”
2. “Working with JAX-RPC handlers and clients” on page 1029.
3. “Working with mediations” on page 1039.
4. “Creating a new UDDI reference” on page 1040.

Creating a new endpoint listener configuration

An endpoint listener is the point (address) at which messages for an inbound service are received. The endpoint listeners that are supplied with WebSphere Application Server support SOAP over HTTP and SOAP over JMS bindings.

Before you begin

For every server that is to host an endpoint listener, you must install and configure a Service Data Objects (SDO) repository on the server.

Note: For WebSphere Application Server Version 6.0.x, you also had to manually install the endpoint listener application before you configured an endpoint listener. For later versions of WebSphere Application Server the endpoint listener application is installed automatically.

If you want to change the default HTTP endpoint listener security role, do so before you configure the SOAP over HTTP endpoint listener.

Before you configure a SOAP over JMS endpoint listener, configure the associated JMS resources.

You can set up separate endpoint listeners for inbound and outbound requests. For more information, see Endpoint listeners and inbound ports: Entry points to the service integration bus.

Decide on which method you want to use to configure these resources. You can create a new endpoint listener configuration by using the administrative console as described in this task, or by using the createSIBWSEndpointListener command.

Note: If you want to create an endpoint listener configuration for your own endpoint listener application, rather than for one of the listeners that is supplied with WebSphere Application Server, you can only do so by using the wsadmin tool.

About this task

A request arrives at an endpoint listener. It is passed to an inbound port (at which point security and JAX-RPC handler lists can be applied) then sent on to the service destination. Responses follow the same path in reverse.

To configure a new endpoint listener for use with an inbound service is a three-stage process:

1. Configure the listener for a specific application server (this task).
2. Connect the listener to one or more service integration buses (the final part of task “Modifying an existing endpoint listener configuration” on page 1026).
3. Configure an inbound service on the same bus to use the listener (part of the task “Making an internally-hosted service available as a Web service” on page 1009).

To use the administrative console to configure an endpoint listener, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click one of the following paths:
   - **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Endpoint listeners**
   - **Servers** → **Clusters** → **WebSphere application server clusters** → ***cluster_name*** → **Endpoint listeners**

   The endpoint listeners collection form is displayed.
3. Click **New**. The New endpoint listener wizard is displayed.
4. Use the wizard to create the new endpoint listener configuration by completing the following steps. For more information about the properties that you set with the wizard, see Endpoint listeners [Settings]. If you want to configure an endpoint listener with values that match those used for an endpoint listener supplied with WebSphere Application Server Version 6.0, see “Endpoint listener configuration details for Version 6.0.x compatibility” on page 1022.
   a. Select listener name and binding type.
      Endpoint listener name
      Type the name by which the endpoint listener is known.
      Binding type
      Select the type of binding that this endpoint listener supports. For example, for a SOAP over HTTP endpoint listener select `SOAP/HTTP`.
   b. Optional: Configure JMS settings.

This panel is only displayed if you selected a JMS binding in the previous panel.

By default, the wizard assumes that you are deploying your endpoint listener application to use a JMS activation specification with the default messaging provider. However, you can choose to deploy your endpoint listener application to use a listener port or another JMS provider (for example WebSphere MQ).

Select from the drop-down lists the **listener port**, or the **activation specification** and **queue connection factory** that you have previously configured as described in "Configuring JMS resources for the synchronous SOAP over JMS endpoint listener" on page 1025.

c. Configure required URLs. Configure Web addresses for the application root and the WSDL serving root. You can either select pre-configured addresses based on the known virtual hosts, or create new values.

URL root

Select or type the address at which external clients access the endpoint listener endpoint. The URL root is the context root of the endpoint listener application, and provides the root of the Web address that is used to build the endpoint addresses within WSDL files to direct requesters to this endpoint listener.

If external clients access the endpoint listener through an HTTP server or server cluster, using default port 80, then specify the HTTP server name and no port number. For example (for SOAP over HTTP endpoint listener 1):
`http://www.yourcompany.com/wsgwsoaphttp1`
However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then specify the application server host name and port number. For example (for SOAP over HTTP endpoint listener 1):
`http://your.server.name:9080/wsgwsoaphttp1`

WSDL serving HTTP URL root

Type the root of the Web address for the WSDL files of the inbound services that are available at this endpoint listener. This address comprises the root of the HTTP address at which external clients access your endpoint listener application, followed by `/sibws`.

If external clients access the endpoint listener through an HTTP server or server cluster, typically using default port 80, then this URL root includes the HTTP server name and no port number. For example:

`http://www.yourcompany.com/sibws`

However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then this URL root includes the application server host name and port number. For example:

`http://your.server.name:9080/sibws`

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service. To get the location details for a given inbound service WSDL file, publish the WSDL file to a zip file as described in "Modifying an existing inbound service configuration" on page 1011, then look up the location within the exported WSDL file.

5. Click **Finish**.

Results

If the processing completes successfully, the list of endpoint listeners is updated to include the new endpoint listener. Otherwise, an error message is displayed.

What to do next

You are now ready to connect one or more service integration buses to this endpoint listener as described in “Modifying an existing endpoint listener configuration” on page 1026, then to select this endpoint listener for use with an inbound service as described in “Making an internally-hosted service available as a Web service” on page 1009.

Endpoint listener configuration details for Version 6.0.x compatibility:

In WebSphere Application Server Version 6.0.x, four endpoint listeners were supplied that are configured as described in this topic. In the current version of the product you can configure any number of endpoint listeners with values of your own choosing, including (if you wish) the values given in this topic.

When you create an endpoint listener configuration for a listener that is supplied with WebSphere Application Server Version 6.0.x, you provide the following configuration details:

Endpoint listener name
The name by which the endpoint listener is known. For example, for SOAP over HTTP endpoint listener 1 the endpoint listener name is `SOAPHTTPChannel1`.

URL root
The address at which external clients access the endpoint listener endpoint. The URL root is the context root of the endpoint listener application, and provides the root of the Web address that is used to build the endpoint addresses within WSDL files to direct requesters to this endpoint listener.

WSDL serving HTTP URL root
The root of the Web address for the WSDL files of the inbound services that are available at this endpoint listener. This address comprises the root of the HTTP address at which external clients access your endpoint listener application, followed by `/sibws`.

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service. To get the location details for a given inbound service WSDL file, publish the WSDL file to a zip file as described in “Modifying an existing inbound service configuration” on page 1011, then look up the location within the exported WSDL file.

This topic specifies these configuration details (endpoint listener name, endpoint listener URL root and WSDL serving URL root) for the following endpoint listeners:

- “SOAP over HTTP endpoint listener 1”
- “SOAP over HTTP endpoint listener 2” on page 1023
- “Synchronous SOAP over Java Message Service (JMS) endpoint listener 1” on page 1023
- “Synchronous SOAP over JMS endpoint listener 2” on page 1024

SOAP over HTTP endpoint listener 1

Endpoint listener name
`SOAPHTTPChannel1`.

URL root
The address at which external clients access the endpoint listener endpoint. If external clients access the endpoint listener through an HTTP server or server cluster, using default port 80, then specify the HTTP server name and no port number. For example (for this endpoint listener):

`http://`*`www.yourcompany.com`*`/wsgwsoaphttp1`However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then specify the application server host name and port number. For example (for this endpoint listener):

http://*your.server.name:9080*/wsgwsoaphttp1

WSDL serving HTTP URL root

The root of the HTTP address at which external clients access your endpoint listener application, followed by /sibws. For example:

http://*www.yourcompany.com*/sibws

or

http://*your.server.name:9080*/sibws

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service.

SOAP over HTTP endpoint listener 2

Endpoint listener name

SOAPHTTPChannel2.

URL root

The address at which external clients access the endpoint listener endpoint. If external clients access the endpoint listener through an HTTP server or server cluster, using default port 80, then specify the HTTP server name and no port number. For example (for this endpoint listener):

http://*www.yourcompany.com*/wsgwsoaphttp2However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then specify the application server host name and port number. For example (for this endpoint listener):

http://*your.server.name:9080*/wsgwsoaphttp2

WSDL serving HTTP URL root

The root of the HTTP address at which external clients access your endpoint listener application, followed by /sibws. For example:

http://*www.yourcompany.com*/sibws

or

http://*your.server.name:9080*/sibws

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service.

Synchronous SOAP over Java Message Service (JMS) endpoint listener 1

Endpoint listener name

SOAPJMSChannel1.

URL root

You specify the properties of the synchronous SOAP over JMS endpoint listener 1 endpoint by using the following syntax:

jms:/*queue_or_topic_indicator*?*property_name=property_value* and so on, separating each property using the "&" character.

For example, if you use the default values for queue destination and queue connection factory when you install the synchronous SOAP over JMS endpoint listeners, then the first part of the end point address is:

jms:/queue?destination=jms/SOAPJMSQueue1&connectionFactory=jms/SOAPJMSFactory1

For each of the synchronous SOAP over JMS endpoint listeners, here is the full list of properties that you can specify in the endpoint address:

Property name	Property description
Destination-related Properties (required)	
connectionFactory	The JNDI name of the queue or topic connection factory.
destination	The JNDI name of the destination queue or topic.
JNDI-related Properties (optional)	
initialContextFactory	The name of the initial context factory to use (this is mapped to the java.naming.factory.initial property).
jndiProviderURL	The JNDI provider Web address (this is mapped to the java.naming.provider.url property).
JMS-related Properties (optional)	
deliveryMode	An indication as to whether or not the request message is persistent. The valid values are 1 (non persistent) and 2 (persistent). The default value is 1.
timeToLive	The lifetime (in milliseconds) of the request message. A value of 0 indicates an infinite lifetime.
priority	The JMS priority associated with the request message. Valid values are 0 - 9. The default value is 4.
userid	The User ID that is used to gain access to the connection factory.
password	The password that is used to gain access to the connection factory.

WSDL serving HTTP URL root

The root of the HTTP address at which external clients access your endpoint listener application, followed by /SIBWS.

If external clients access the endpoint listener through an HTTP server or server cluster, typically using default port 80, then this URL root includes the HTTP server name and no port number. For example:

http://*www.yourcompany.com*/sibws

However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then this URL root includes the application server host name and port number. For example:

http://*your.server.name:9080*/sibws

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service.

Synchronous SOAP over JMS endpoint listener 2

Endpoint listener name

SOAPJMSChannel2.

URL root

You specify the properties of the synchronous SOAP over JMS endpoint listener 2 endpoint by using the following syntax:

jms:/*queue_or_topic_indicator*?*property_name*=*property_value* and so on, separating each property using the "&" character.

For example, if you use the default values for queue destination and queue connection factory when you install the synchronous SOAP over JMS endpoint listeners, then the first part of the end point address is:

```
jms:/queue?destination=jms/SOAPJMSQueue2&connectionFactory=jms/SOAPJMSFactory2
```

For the full list of properties that can be specified in the endpoint address for synchronous SOAP over JMS endpoint listener 2, see the list of properties detailed above for Synchronous SOAP over JMS endpoint listener 1.

WSDL serving HTTP URL root

The root of the HTTP address at which external clients access your endpoint listener application, followed by `/SIBWS`.

If external clients access the endpoint listener through an HTTP server or server cluster, typically using default port 80, then this URL root includes the HTTP server name and no port number. For example:

```
http://www.yourcompany.com/sibws
```

However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then this URL root includes the application server host name and port number. For example:

```
http://your.server.name:9080/sibws
```

Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service.

Configuring JMS resources for the synchronous SOAP over JMS endpoint listener:

Configure the synchronous SOAP over Java Message Service (JMS) endpoint listeners to use a JMS provider - either the default messaging provider, or another provider such as WebSphere MQ - to pass SOAP messages over JMS.

About this task

To install and configure the JMS provider for the synchronous SOAP over JMS endpoint listener, complete the following steps.

1. Optional: If you have not already done so, choose and configure a JMS messaging provider.
2. Use the administrative console to create and configure queue connection factories and queue destinations as described, for the most common JMS providers, in topics available from within Choosing a messaging provider. Create a connection factory and a queue for each endpoint listener that you plan to configure. For example if you plan to configure both of the SOAP over JMS endpoint listeners that are supplied with WebSphere Application Server, create two connection factories (one for each endpoint listener) and two queues. The JMS resources and JNDI names that the supplied SOAP over JMS endpoint listeners expect by default are provided in the following table. If you use different resources and names in this step, then change the defaults when you subsequently configure the endpoint listener.

JMS resource	default JNDI name (endpoint listener 1)	default JNDI name (endpoint listener 2)	queue name (endpoint listener 1)	queue name (endpoint listener 2)
JMS queue connection factory	jms/SOAPJMSFactory1	jms/SOAPJMSFactory2	Not required	Not required
JMS queue	jms/SOAPJMSQueue1	jms/SOAPJMSQueue2	User defined (for example: SOAPJMSDestQueue1)	User defined (for example: SOAPJMSDestQueue2)

3. Configure the underlying destination for each JMS queue.

If you are using the default messaging provider, use the administrative console to add the two new queue names specified in the previous table as destinations for your application server as described in Creating a queue for point-to-point messaging. The identifier for the destination should match that defined by the user as the queue name in the previous table.

Otherwise, configure these destinations as described in the documentation for your JMS provider.

4. Configure the deployment details for the application.

   If you are using activation specifications with the default messaging provider, use the administrative console to create and configure the activation specifications as described in “Configuring a JMS activation specification for the default messaging provider” on page 1591. Create two activation specifications, one for each endpoint listener. The default JMS resources and associated names that the synchronous SOAP over JMS endpoint listeners expect are provided in the following table. However, you can use any JNDI name for the activation specification, provided that the EAR file has the same JNDI reference in the administrative console “Binding enterprise beans to listener port names or activation specification JNDI names” panel. If you use different resources and names in this step, change the defaults when you subsequently configure the endpoint listener. You must also stop then restart the application server.

JMS resource	default JNDI name (endpoint listener 1)	default JNDI name (endpoint listener 2)	destination JNDI name (endpoint listener 1)	destination JNDI name (endpoint listener 2)
activation specification	eis/SOAPJMSChannel1	eis/SOAPJMSChannel2	jms/SOAPJMSQueue1	jms/SOAPJMSQueue2

   If you are using listener ports with any supported JMS provider (including the default messaging provider), use the administrative console to create and configure the listener ports in the message listener service as described in Adding a new listener port. Create two listener ports (one for each endpoint listener). The default JMS resources and associated names that the supplied SOAP over JMS endpoint listeners expect are provided in the following table. If you use different resources and names in this step, then change the defaults when you subsequently configure the endpoint listener.

JMS resource	default name (for use with SOAP over JMS endpoint listener 1)	default name (for use with SOAP over JMS endpoint listener 2)
listener port	SOAPJMSPort1	SOAPJMSPort2
connection factory	jms/SOAPJMSFactory1	jms/SOAPJMSFactory2
destination	jms/SOAPJMSQueue1	jms/SOAPJMSQueue2

5. Save your changes to the master configuration.
6. Bind the JMS resources by stopping then restarting the application server.

What to do next

You are now ready to create a new SOAP over JMS endpoint listener configuration.

Modifying an existing endpoint listener configuration:

Modify the additional properties for an endpoint listener that has been configured for use with inbound services.

Before you begin

You cannot use this task to modify the general properties of an existing endpoint listener. The reason for this is that, when you create a new endpoint listener, an associated endpoint listener application is automatically installed that uses the same general property values. Therefore if you subsequently change the general properties, you break the link between the configuration and the underlying application. If you need to change the general properties, you should delete and recreate the endpoint listener configuration.

About this task

To list the endpoint listeners, and to view and modify their configuration details, complete the following steps.

1. Start the administrative console.
2. In the navigation pane, click one of the following paths:
   - **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Endpoint listeners**
   - **Servers** → **Clusters** → **WebSphere application server clusters** → ***cluster_name*** → **Endpoint listeners**

   A list of endpoint listeners is displayed in an endpoint listener collection form.
3. Click the name of an endpoint listener in the list. The current endpoint listener settings for this endpoint listener are displayed.
4. View the following general properties. If you are configuring an endpoint listener that is supplied with WebSphere Application Server, the required values are given in Endpoint listener configuration details.

   Name View the name by which the endpoint listener is known. If this is your own endpoint listener, rather than one that is supplied with WebSphere Application Server, then this name must match the name given in the endpoint listener application that you have installed (that is, the *display name* of the endpoint module within the endpoint application EAR file).

   DescriptionDescription
   View the (optional) description of the endpoint listener.

   URL root
   View the address at which external clients access the endpoint listener endpoint. The URL root is the context root of the endpoint listener application, and provides the root of the Web address that is used to build the endpoint addresses within WSDL files to direct requesters to this endpoint listener.

   If external clients access the endpoint listener through an HTTP server or server cluster, using default port 80, then specify the HTTP server name and no port number. For example (for SOAP over HTTP endpoint listener 1):
   `http://www.yourcompany.com/wsgwsoaphttp1`
   However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then specify the application server host name and port number. For example (for SOAP over HTTP endpoint listener 1):
   `http://your.server.name:9080/wsgwsoaphttp1`

   WSDL serving HTTP URL root
   View the root of the Web address for the WSDL files of the inbound services that are available at this endpoint listener. This address comprises the root of the HTTP address at which external clients access your endpoint listener application, followed by `/sibws`.

   If external clients access the endpoint listener through an HTTP server or server cluster, typically using default port 80, then this URL root includes the HTTP server name and no port number. For example:

`http://www.yourcompany.com/sibws`

   However, if you allow external clients to connect direct to your application server (for example in a development or test environment) then this URL root includes the application server host name and port number. For example:
   `http://your.server.name:9080/sibws`

   Note: The WSDL serving HTTP URL root is only used internally by other components of WebSphere Application Server (notably the IBM UDDI registry). For all other uses, you access the WSDL file through the endpoint listener endpoint for the inbound service. To get the location details for a given inbound service WSDL file, publish the WSDL file to a zip file as described in “Modifying an existing inbound service configuration” on page 1011, then look up the location within the exported WSDL file.

5. Under the additional properties heading, click **Connection properties**. A list of all the service integration buses that are currently connected to this endpoint listener is displayed in a service integration bus connection properties collection form. Add, amend or delete buses in the list of currently-connected buses. To add a new bus, complete the following steps:
   a. Click **New**. The service integration bus connection properties settings form is displayed.
   b. Under the general properties heading, choose an available service integration bus from the selection list. The bus is selected and the additional properties for the bus are displayed.

      Note: Under the connection properties for the bus there are **Custom properties**. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value. You use custom properties to define the manner in which the endpoint listener connects to this bus. Included in this set is property name `com.ibm.ws.sib.webservices.replyDestination`, which defines the reply destination name used by the endpoint listener. Do not modify or remove this property, which is set automatically when the service integration bus is associated with the endpoint listener.
6. Under the additional properties heading, click **Associated application**. Details for the application that handles the requests for this endpoint listener are displayed in an enterprise application settings form.
7. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of service integration buses that are connected to this endpoint listener is updated, and the list of endpoint listeners is redisplayed. Otherwise, an error message is displayed.

What to do next

You are now ready to select this endpoint listener for use with an inbound service as described in “Making an internally-hosted service available as a Web service” on page 1009.

Deleting endpoint listener configurations:
Before you begin

You cannot delete an endpoint listener that has inbound ports associated with it. If you try to do this, an error is generated. Before you can delete an endpoint listener, you must either delete each associated inbound service or remove each associated port from the port list for the inbound service.

Decide on which method you want to use to configure these resources. You can delete an endpoint listener configuration by using the administrative console as described in this task, or by using the deleteSIBWSEndpointListener command.

About this task

To remove one or more endpoint listener configurations, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Endpoint listeners**. A list of endpoint listeners is displayed in an endpoint listener collection form.
3. Select the check box for every endpoint listener configuration that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of endpoint listeners is updated. Otherwise, an error message is displayed.

Working with JAX-RPC handlers and clients

The Java API for XML-based remote procedure calls (JAX-RPC) provides you with a standard way of developing interoperable and portable Web services. You can use JAX-RPC handlers, handler lists and client applications with your service integration bus-enabled Web services.

About this task

There are two main elements of JAX-RPC that you can use directly with the service integration bus:

- JAX-RPC handlers and handler lists.
- JAX-RPC client applications.

A JAX-RPC handler is a Java class that performs a range of handling tasks.For example: logging messages, or transforming their contents, or terminating an incoming request. To create a JAX-RPC handler, you can use a tool such as IBM Rational Application Developer. To enable handlers to perform more complex operations, you chain them together into handler lists. You associate each handler list with one or more ports, so that the handler list can monitor activity at the port, and take appropriate action depending upon the sender and content of each message that passes through the port.

JAX-RPC client applications send and receive Web service request and response messages. JAX-RPC client applications that use the IBM JAX-RPC run-time environment can do this in a number of different ways, depending on the bindings in the WSDL document that they are developed against, and the configuration data that is used at run time.

Detailed instructions on how to configure JAX-RPC handlers, handler lists and client applications for use with the service integration bus are provided in the following topics:

- “Creating a new JAX-RPC handler configuration.”
- “Creating a new JAX-RPC handler list” on page 1033.
- “Sending Web service messages directly over the bus from a JAX-RPC client” on page 1036.
- “Implementing JAX-RPC handlers to access SDO messages” on page 1038.

Creating a new JAX-RPC handler configuration:

Create a JAX-RPC handler configuration for use, as part of a handler list, with service integration bus-deployed Web services. Handlers monitor messages at ports, and take appropriate action depending upon the sender and content of each message.

Before you begin

This task assumes that you have already created your handler. You can do this using IBM Rational Application Developer or a similar tool. For more information, see the IBM developerWorks article Support for J2EE Web Services in WebSphere Studio Application Developer V5.1 -- Part 3: JAX-RPC Handlers.

You must also make the handler class available to the server that hosts the port for the service that you want to monitor, as detailed in “Loading JAX-RPC handler classes” on page 1030.

About this task

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. To make WebSphere Application Server aware of your handler, and to make the handler

available for inclusion in one or more handler lists, you use the administrative console to create a new handler configuration.

1. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handlers**. The JAX-RPC handlers collection form is displayed.
2. Click **New**. The JAX-RPC handlers settings form is displayed.
3. Type the following general properties:
   Name Type the name by which the handler is known.

   This name must be unique, and it must obey the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   For example `TestHandler`.

   Description
   Type the (optional) description of the handler.

   Class name
   Type the name of the class that is to be instantiated. For example `com.ibm.jaxrpc.handler.TestHandler`.

   Note: You can configure multiple instances of a handler by creating each instance with a different handler name, and pointing to the same handler class.
4. Click **OK**. The general properties for this item are saved, and the additional properties options are made available.
5. Type the following additional properties:
   SOAP roles
   Add SOAP actor definitions to the list of SOAP roles in which this handler acts. For more information, see the SOAP specification.

   JAX-RPC headers
   Add JAX-RPC header definitions (Namespace URI and Local part) to the list of JAX-RPC headers against which this handler operates. JAX-RPC headers are SOAP headers that are processed by a JAX-RPC handler.

   Custom properties
   Add custom properties (name/value pairs, where the name is a property key and the value is a string value that can be used to set internal system configuration properties).
6. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handlers is updated to include the new handler. Otherwise, an error message is displayed.

What to do next

To use this handler, add it to a handler list as described in Creating a new JAX-RPC handler list or Modifying an existing JAX-RPC handler list.

Loading JAX-RPC handler classes:

A JAX-RPC handler interacts with messages as they pass into and out of the service integration bus, therefore you make the handler class available to the server or cluster that hosts the inbound or outbound port for the service that you want to monitor.

Before you begin

This task assumes that you have already created your handler. You can do this using IBM Rational Application Developer or a similar tool. For more information, see the IBM developerWorks article Support for J2EE Web Services in WebSphere Studio Application Developer V5.1 -- Part 3: JAX-RPC Handlers.

About this task

Before you can configure your JAX-RPC handler for use with service integration bus-deployed Web services, you must make the handler class available. If you want to monitor an inbound port, make the handler class available to the server on which the endpoint listener for that port is located. If you want to monitor an outbound port, make the handler class available to the server on which the outbound port destination is localized.

To make the handler class available to the server that hosts the port that you want to monitor, you create a shared library for the class then add the shared library to the class loader for the server.

1. Package the class file for your handler as a JAR file, then copy the JAR file into a convenient directory.

   Make the handler class available to the application server in one of the following ways:

   - Copy the individual class file into a directory structure under *app_server_root*/classes that matches the package name of the class, where *app_server_root* is the root directory for the installation of WebSphere Application Server. For example a handler class com.ibm.jaxrpc.handler.TestHandler is copied into the *app_server_root*/classes/com/ibm/jaxrpc/handler directory.
   - Package the class files for all your handlers as a JAR file, then copy it into the *app_server_root*/lib/app directory.

2. Start the administrative console.
3. Create a shared library for the JAR file.
   a. Navigate to **Environment** → **Shared libraries**.
   b. Set the scope at which you want the new library to be visible, then click **New**.
   c. Give the new library a name.
   d. Set the class path to the directory and file name for your handler JAR file.
   e. Save your changes to the master configuration.

   For more information, see Creating shared libraries.
4. Create a class loader for the server on which you want to make the JAR file available.
   a. Navigate to **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server Infrastructure] Java and Process Management** → **Class loader**.
   b. Click **New**.
   c. Click **OK**.
   d. Save your changes to the master configuration.

   For more information, see Configuring class loaders of a server.
5. Add the shared library to the class loader for the server.
   a. Navigate to **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server Infrastructure] Java and Process Management** → **Class loader** → ***class_loader_name*** → **[Additional Properties] Shared library references**.
   b. Click **Add**.
   c. Click on the name of your new library, then click **OK**.
   d. Save your changes to the master configuration.

   For more information, see Associating shared libraries with servers.

What to do next

You are now ready to configure your handler for use (as part of a handler list) with service integration bus-enabled Web services.

Modifying an existing JAX-RPC handler configuration:

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. Handlers monitor messages at ports, and take appropriate action depending upon the sender and content of each message. Modify the configuration details for a JAX-RPC handler that has been configured for use, as part of a handler list, with service integration bus-deployed Web services.

Before you begin

If you modify a handler class but do not change the class name, you do not need to modify the handler configuration as described in this topic. You just need to stop then restart the servers that host the ports that this handler monitors.

About this task

To list the handlers, and to view and modify their configuration details, complete the following steps.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handlers**. A list of handlers is displayed in a JAX-RPC handlers collection form.
3. Click the name of a handler in the list. The current JAX-RPC handlers settings for this handler are displayed.
4. Modify the following general properties:
   Name Modify the name of the handler.

   This name must be unique, and it must obey the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   Note: When you change a handler name, the system looks up all objects that refer to it and updates the name.

   Description
   Modify the (optional) description of the handler.

   Class name
   Change the name of the class that is to be instantiated. If you change the class name, you must also make the new class available to the servers that host the ports that this handler monitors, as detailed in "Loading JAX-RPC handler classes" on page 1030.
5. Modify the following additional properties:
   SOAP roles
   Add, modify or remove SOAP actor definitions from the list of SOAP roles in which this handler acts. For more information, see the SOAP specification.

   JAX-RPC headers
   Add, modify or remove JAX-RPC header definitions (Namespace URI and Local part) from the list of JAX-RPC headers against which this handler operates. JAX-RPC headers are SOAP headers that are processed by a JAX-RPC handler.

   Custom properties
   Add, modify or remove custom properties (name/value pairs, where the name is a property key and the value is a string value that can be used to set internal system configuration properties).

6. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handlers is redisplayed. Otherwise, an error message is displayed.

Deleting JAX-RPC handler configurations:

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. Delete JAX-RPC handlers that are configured for use (as part of a handler list) with service integration bus-deployed Web services.

About this task

When you remove a handler that is currently used by one or more Web services on a service integration bus, the system removes the handler from the handler lists for each associated Web service.

To remove one or more handlers that are currently configured for a service integration bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handlers**. A list of handlers is displayed in a JAX-RPC handlers collection form.
2. Select the check box for every handler that you want to remove.
3. Click **Delete**.

Results

If the processing completes successfully, the list of handlers is updated. Otherwise, an error message is displayed.

Creating a new JAX-RPC handler list:

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. Create a JAX-RPC handler list for use with service integration bus-enabled Web services.

Before you begin

You can only add previously-configured handlers to a handler list. To configure a handler, see Creating a new JAX-RPC handler configuration.

About this task

Handlers monitor messages at ports, and take appropriate action depending upon the sender and content of each message. To enable handlers to perform more complex operations, you chain them together into handler lists. The approach taken in WebSphere Application Server is to apply handler lists (rather than individual handlers) at the ports, where each handler list contains one or more handlers.

To create a new JAX-RPC handler list, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handler Lists**. The JAX-RPC handler lists collection form is displayed.
2. Click **New**. The JAX-RPC handler lists settings form is displayed.
3. Type the following general properties:

Name Type the name by which the handler list is known.

This name must be unique, and it must obey the following syntax rules:
- It must not start with "." (a period).
- It must not start or end with a space.
- It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

For example `TestList`.

Description
Type the (optional) description of the handler list.

JAX-RPC handlers
In the JAX-RPC handlers pane, complete the following steps:
a. Select one or more handlers from the list of available JAX-RPC handlers, then click **Add** to move the selected handlers into the list of handlers for this JAX-RPC handler list.
b. Select a handler in the list of handlers for this JAX-RPC handler list, then click **Up** or **Down** to change the position of the handler within the list.

Handlers are applied in the sequence in which they appear in the handler list.

Note: If you click **Reset**, only the **Name** and **Description** fields are reset to their state when the form was first loaded. The two lists of available and assigned handlers are not reset.

4. Click **OK**. The general properties for this item are saved.
5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handler lists is updated to include the new handler list. Otherwise, an error message is displayed.

What to do next

To use this handler list, select it for use with a Web service as described in "Modifying an existing inbound service configuration" on page 1011 or "Modifying an existing outbound service configuration" on page 1016.

Modifying an existing JAX-RPC handler list:

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. Modify the configuration details for a JAX-RPC handler list that has been configured for use with service integration bus-deployed Web services.

Before you begin

You can only add previously-configured handlers to a handler list. To configure a handler, see Creating a new JAX-RPC handler configuration.

About this task

Handlers monitor messages at ports, and take appropriate action depending upon the sender and content of each message. To enable handlers to perform more complex operations, you chain them together into handler lists. The approach taken in WebSphere Application Server is to apply handler lists (rather than individual handlers) at the ports, where each handler list contains one or more handlers.

To list the existing handler lists, and to view and modify their configuration details, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handler Lists**. A list of all the handler lists is displayed in a JAX-RPC handler lists collection form.
2. Click the name of a handler list in the list. The current JAX-RPC handler lists settings for this handler are displayed.
3. Modify the following general properties:
   Name Modify the name of the handler list.

   This name must be unique, and it must obey the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   When you change a handler list name, the system looks up all objects that refer to it and updates the name.

   Description
   Modify the (optional) description of the handler list.

   JAX-RPC handlers
   In the JAX-RPC handlers pane, complete the following steps:
   a. Select one or more previously-configured handlers from either the list of available JAX-RPC handlers or the list of handlers for this JAX-RPC handler list, then click **Add** or **Remove** to modify the list of handlers for this JAX-RPC handler list.
   b. Select a handler in the list of handlers for this JAX-RPC handler list, then click **Up** or **Down** to change the position of the handler within the list.

   Handlers are applied in the sequence in which they appear in the handler list.

   Note: If you click **Reset**, only the **Name** and **Description** fields are reset to their state when the form was first loaded. The two lists of available and assigned handlers are not reset.
4. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handler lists is redisplayed. Otherwise, an error message is displayed.

What to do next

To use this handler list, select it for use with a Web service as described in "Modifying an existing inbound service configuration" on page 1011 or "Modifying an existing outbound service configuration" on page 1016.

Deleting JAX-RPC handler lists:

A Java API for XML-based remote procedure calls (JAX-RPC) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. Delete JAX-RPC handler lists that are configured for use with service integration bus-deployed Web services.

About this task

When you remove a handler list that is currently used by one or more Web services on a service integration bus, the system removes the handler list for each associated Web service.

To remove one or more handler lists, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Web services** → **JAX-RPC Handler Lists**. A list of handler lists is displayed in a JAX-RPC handler lists collection form.

2. Select the check box for every handler list that you want to remove.
3. Click **Delete**.

Results

If the processing completes successfully, the list of handler lists is updated. Otherwise, an error message is displayed.

Sending Web service messages directly over the bus from a JAX-RPC client:
About this task

Java API for XML-based remote procedure calls (JAX-RPC) client applications send and receive Web service request and response messages. JAX-RPC client applications using the IBM JAX-RPC run-time environment can do this in a number of different ways, depending on the bindings in the WSDL document that they are developed against, and the configuration data that is used at run time.

For an introduction to basic JAX-RPC programming concepts, including the JAX-RPC client and server programming models, see Getting Started with JAX-RPC.

If you want to use a JAX-RPC client to send messages over the service integration bus, you have two choices:

- Use a SOAP binding (SOAP over HTTP or SOAP over JMS), and pass messages indirectly through an endpoint listener to an inbound service. You would do this if you had SOAP-specific JAX-RPC handlers that must run in the client application context.
- Pass messages directly into the service integration bus at a destination by “retargeting” the JAX-RPC client application as described in this topic.

  Note: There are currently limitations regarding the Java types used by services that are retargeted through a JAX-RPC client application.

Retargeting involves setting the following two values into the client application deployment descriptor, or specifying them dynamically at run time from within the client application:

- The *binding namespace* is set to indicate that the client uses the messaging bus directly.
- The *endpoint address* is set to include the particular destination and (optionally) the format of messages that the client uses.

The destination also needs to be configured so that it knows the port type of messages that the JAX-RPC client is using. There are two ways to achieve this:

- Create an outbound service. An outbound service represents an externally-provided Web service. In this case, requests from the JAX-RPC client pass through the service destination and are then sent on to the service provider defined by the outbound service configuration.
- Create an inbound service. An inbound service represents a service provided somewhere within or beyond the messaging bus. You can create an inbound service on any existing destination. The creation of an inbound service associates a WSDL port type with the destination. When retargeting to a destination with an inbound service, the client application needs to specify both the destination name and inbound service name, because it is possible to configure more than one inbound service against a single destination. In this case, requests from the JAX-RPC client pass through the destination and then onwards through the service integration bus depending on routing that is done at the initial destination.

To have Web service messages sent directly to a destination using a JAX-RPC client, complete the following steps:

1. Create the JAX-RPC client application.
2. Create the outbound service or inbound service with which you want the JAX-RPC client application to exchange messages.

3. Use the administrative console to access the port information for your JAX-RPC client application, as described in Configuring Web service client bindings and Web services client port information.
4. Override the default SOAP binding for your JAX-RPC client application. Change the binding namespace to `http://www.ibm.com/ns/2004/02/wsdl/mp/sib`
5. Override the endpoint that your JAX-RPC client application uses to send Web service requests. The new endpoint should use the sib: URL syntax and include either the outbound service destination name, or both the inbound service name and its corresponding destination name.

What to do next

After you change the *binding namespace*, any JAX-RPC handler lists that were configured for the retargeted port are ignored. For clients that are developed against WSDL with a SOAP binding, retargeting directly to the bus causes the handlers to be ignored. However if the client is developed against the non-bound WSDL for the service, retargeting to the bus is not considered to be changing the binding namespace, and so the handler information is retained. In this case the JAX-RPC handlers are called with the SDOMessageContext subclass.

Associated reference information:

- "sib: URL syntax"

sib: URL syntax:

The `sib:` URL has the following syntax:

`sib:/[destination|path]?property_1=value_1&property_2=value_2&...`

where:

- Square brackets ("[ ]") indicate that a parameter is optional.
- Transport type is `sib:`, followed by either `/destination` to specify destination type or `/path` to specify a forward routing path, followed by a "query string" that contains one or more properties. The permitted properties are described in the subsequent sections of this topic.

Required properties

The following properties are required. They are used to specify the destination for the request.

Note: All destination names must be fully-qualified. That is, they must include the name of the service integration bus as well as the destination name itself. Use the syntax *bus:destination*. If a bus or destination name contains a colon or comma, wrap the name in double quotation marks (""). If it contains a double quotation mark, repeat the quotation mark.

destinationName
The destination name.

path The forward routing path, in the form of a sequence of destination names separated by commas.

replyDestinationName
The name of the destination to be used for the reply.

inboundService
The name of the inbound service that identifies the specific attachment that the requester application uses. You can omit this value if the destination is a service destination with an associated outbound service configuration, because in that case the requester is attaching to the outbound service through the service destination.

timeout
The time the requester waits for a response. The default value is 60 seconds. A zero value indicates an unlimited wait.

Service integration technologies-related properties

The following properties are optional. If you do not specify a value for a property, then the default value is used. For more information regarding the permitted values for these properties, see the generated API information for the SIMessage interface.

reliability
The reliability of the request message.

timeToLive
The amount of time (in milliseconds) before the request times out. A zero value indicates that the request never times out.

Note: The **timeout** property (see the required properties) is the time delay after which the requester application blocks the application thread that is waiting for a response to a request and response operation. The **time to live** and **replyTimeToLive** optional properties indicate how long the request and reply messages should be processed by the messaging engines. This does not include the processing time at the service implementation. You would therefore usually set the timeout to be the sum of the request and response times to live, plus some amount for the service processing time.

priority
The priority of the request message.

user
The user ID required to access the request destination.

password
The password required to access the request destination.

replyReliability
The reliability of the reply message.

replyTimeToLive
The amount of time (in milliseconds) before the reply times out. A zero value indicates that the reply never times out.

replyPriority
The priority of the reply message.

Other properties

You can also include user-defined properties in the URL. These properties must be named with a `user.` prefix. For example:

```
sib:/destination?destinationName=myBus:myDestination & reliability=assured & user.customData=XYZ
```

After the request is sent, the URL itself is available within the message properties, named `inbound.url`.

Implementing JAX-RPC handlers to access SDO messages:

JAX-RPC handlers are invoked during processing of request and response messages. For messages that are exchanged using the SOAP protocol, each JAX-RPC handler is passed a SOAP-specific `MessageContext` object. For other protocols, the IBM Web services runtime environment passes a `MessageContext` object that provides a Service Data Objects view of the message. Service Data Objects (SDO) is an open standard for enabling applications to handle data from different data sources in a uniform way, as DataGraphs.

If the JAX-RPC handler only deals with message context properties, then it does not need to be aware of the particular subclass of `MessageContext` that it is given, because the context property methods are defined by the `MessageContext` interface itself. If the handler needs to process information contained within

the message, then it must be coded to work with the required subclasses. It is recommended that your JAX-RPC handlers test whether the `MessageContext` is an instance of the required subclass.

A JAX-RPC handler is given an SDO-specific `MessageContext` object (an instance of the `com.ibm.websphere.webservices.handler.sdo.SDOMessageContext` class) rather than the SOAP-specific `MessageContext` object in the following cases:

- A JAX-RPC client or outbound invocation from the service integration bus invokes a service using the EJB binding.
- A JAX-RPC client is developed against a non-bound WSDL and is retargeted to a destination in the service integration bus.

The `SDOMessageContext` class provides methods to get and set the `com.ibm.websphere.sdo.SDOMessage` instance that represents the actual message being processed. The `SDOMessage` in turn has a method to access the SDO DataGraph object that holds the message content as SDO DataObjects.

A JAX-RPC handler can modify the SDO DataGraph contents, but it cannot change the format or schema of the message.

Here is an example of the code that is used to access the SDO DataGraph from the `MessageContext` object in a JAX-RPC handler `handleRequest` method:

```
public boolean handleRequest(MessageContext messageContext) {

  // Convert the MessageContext into an SDOMessageContext
  if( messageContext instanceof SDOMessageContext) {
   SDOMessageContext smc = (SDOMessageContext)messageContext;

   // Retrieve the message
   SDOMessage message = smc.getSDOMessage();

   // Get the root object in the SDO DataGraph
   DataGraph graph = message.getDataGraph();
   DataObject content = graph.getRootObject();

   // Now do something with the message content.....
  }
  return true;
 }
```

Working with mediations

Use a mediation to change the content of a message, or the way in which a message is handled.

Before you begin

For an introduction to using mediations with the service integration bus, see Learning about mediations.

About this task

A mediation is an application that contains a mediation handler. You associate a mediation with a service integration bus destination, and the mediation acts on messages that pass through the destination. The action taken by a mediation depends upon the specific instructions you give in the mediation handler. For example, you can use a mediation to change the contents of a message, or to choose a particular forward route for a message.

To write a mediation application that contains a mediation handler, install it into WebSphere Application Server and associate it with a bus destination, complete the following steps:

1. Create the mediation application. For examples of how to do this, see:
   - Writing a routing mediation

- Writing a mediation that maps between attachment encoding styles

2. Install the mediation application into WebSphere Application Server.
3. Create the associated mediation object in the bus.

   Note: The **handler list name** is the name you gave to your handler when you defined the handler class as a mediation handler.
4. Mediate the destination. Use the administrative console to access the destination that you want to mediate, then select your mediation from the list of available mediations.

Creating a new UDDI reference

Create a reference (a pointer) to a UDDI registry for use with service integration bus-enabled Web services.

Before you begin

This topic assumes that you have already created a J2C authentication alias for each *Authorized Name* that you want to associate with this UDDI reference. For more information about *Authorized Names*, see UDDI registries: Web service directories that can be referenced by bus-enabled Web services.

About this task

A UDDI reference is a pointer to a UDDI registry. This registry can be a private UDDI registry such as the IBM WebSphere UDDI Registry, or a public UDDI registry. The service integration technologies interact with UDDI registries in two ways:

- When you configure an inbound service, you can create entries for the Web service in one or more UDDI registries.
- When you configure an outbound service, you specify the location of the target WSDL file that describes the Web service. This WSDL file can be located at a URL or through a UDDI registry.

In the UDDI model, Web services are owned by businesses, and businesses are owned by Authorized Names. Each UDDI reference gives access to the Web services that are owned by a single Authorized Name in a single UDDI registry. For more information about how the service integration technologies work with UDDI registries, see UDDI registries: Web service directories that can be referenced by bus-enabled Web services and Publishing a Web service to a UDDI registry. For more general information about UDDI and UDDI registries, see the UDDI community at uddi.org.

To create a new UDDI reference, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **UDDI References**. The UDDI references collection form is displayed.
3. Click **New**. The UDDI references settings form is displayed.
4. Type the following general properties:

   Name Type the name by which the UDDI reference is known. This name must be unique, and it must follow the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   You might need more than one UDDI reference for a given UDDI registry. For more information, see UDDI registries: Web service directories that can be referenced by bus-enabled Web services.

   Description
   Type the (optional) description of the UDDI reference.

   Inquiry URL
   Type the Web address that provides access to this registry for the SOAP inquiry API.

Publish URL

Type the Web address that provides access to this registry for the SOAP publish API.

Authentication alias

Type the J2C authentication alias for the User ID and password of an “Authorized Name” that has update access to this registry.

The alias you enter here must match the user ID of the owner of the corresponding business in the UDDI registry. You can see the owning user ID in UDDI by looking at the business details under the “Authorized Name” field. If the alias you enter here does not match the “Authorized Name” value for the business that owns the service, then the service is not published or found.

If the business has more than one “Authorized Name”, you might want to set up multiple UDDI references (each with a different authentication alias) to the same UDDI registry.

5. Click **OK**.

Results

If the processing completes successfully, the list of UDDI references is updated to include the new UDDI reference. Otherwise, an error message is displayed.

What to do next

To use this UDDI reference, select it when “Making an internally-hosted service available as a Web service” on page 1009 or “Making an externally-hosted Web service available internally” on page 1014.

Modifying an existing UDDI reference:

Modify the configuration details for a UDDI reference that has been configured for use with service integration bus-enabled Web services.

About this task

A UDDI reference is a pointer to a UDDI registry. This registry can be a private UDDI registry such as the IBM WebSphere UDDI Registry, or a public UDDI registry. The service integration technologies interact with UDDI registries in two ways:

- When you configure an inbound service, you can create entries for the Web service in one or more UDDI registries.
- When you configure an outbound service, you specify the location of the target WSDL file that describes the Web service. This WSDL file can be located at a URL or through a UDDI registry.

To list the UDDI references, and to view and modify their configuration details, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **UDDI References**. A list of UDDI references is displayed in a UDDI references collection form.
3. Click the name of a UDDI reference in the list. The current UDDI reference settings for this UDDI reference are displayed.
4. Modify the following general properties:
   Name Modify the name of the UDDI reference.

   This name must be unique, and it must obey the following syntax rules:
   - It must not start with “.” (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

Note: When you change a UDDI reference name, the system looks up all objects that refer to it and updates the name.

Description

Modify the (optional) description of the UDDI reference.

Inquiry URL

Type the new Web address that provides access to this registry for the SOAP inquiry API.

Publish URL

Type the new Web address that provides access to this registry for the SOAP publish API.

Authentication alias

Type the new J2C authentication alias for the User ID and password of an "Authorized Name" that has update access to this registry.

The alias you enter here must match the user ID of the owner of the corresponding business in the UDDI registry. You can see the owning user ID in UDDI by looking at the business details under the "Authorized Name" field. If the alias you enter here does not match the "Authorized Name" value for the business that owns the service, then the service is not published or found.

If the business has more than one "Authorized Name", you might want to set up multiple UDDI references (each with a different authentication alias) to the same UDDI registry.

5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of UDDI references is redisplayed. Otherwise, an error message is displayed.

What to do next

To use this UDDI reference, select it when "Making an internally-hosted service available as a Web service" on page 1009 or "Making an externally-hosted Web service available internally" on page 1014.

Deleting UDDI references:

delete UDDI references that are configured for use with service integration bus-deployed Web services.

About this task

When you remove a UDDI reference that is currently used by one or more Web services on a service integration bus, the system removes the UDDI reference for each associated Web service and makes any necessary updates in the associated UDDI registry.

To remove UDDI references, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **UDDI References**. A list of UDDI references is displayed in a UDDI references collection form.
3. Select the check box for every UDDI reference that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of UDDI references is updated. Otherwise, an error message is displayed.

Publishing a Web service to a UDDI registry:

When you configure an inbound or outbound service, you enable UDDI interaction by associating the service with a UDDI reference, and (depending upon what you are trying to do) either or both of the following pieces of information: The *business key* that identifies the UDDI business category under which you want your service to appear in the UDDI registry, and the service-specific part of the *service key* that the UDDI registry assigns to your service. To help you understand what UDDI business keys and service keys are, and where you find them in a UDDI registry, here is a description of how to publish a Web service to a UDDI registry.

About this task

Service integration technologies interact with UDDI registries as described in UDDI registries: Web service directories that can be referenced by bus-enabled Web services. When you publish a Web service to a UDDI registry, you:

- Specify the type of business that your Web service supports. This usually means choosing an existing business type from a list, but you can also create a new business type. For each type of business there is an associated business key. Service integration bus-enabled Web services use this key, in combination with the service key, to find the Web service in the registry.
- Add a Technical model. Technical models are generic categories. Using these models, a UDDI registry user can search for a type of service, rather than needing to know the access details for a specific service. Bus-enabled Web services interact with UDDI registries at the level of individual Web services, and therefore do not use Technical models.
- Add the Web service. The UDDI registry assigns a service key to your service, and publishes the service. Bus-enabled Web services use this key, in combination with the business key, to find the Web service in the registry.

The following steps describe how you publish a Web service to the IBM WebSphere UDDI Registry. If you are working with a different UDDI registry, then the specific navigation is different but the underlying principles are the same.

1. Specify a business:
   a. To get a list of valid business keys, look up `businesses` in the UDDI registry. Here is an example of a UDDI business key:

      ```
      08A536DC-3482-4E18-BFEC-2E2A23630526
      ```

      .
   b. If you do not find an appropriate existing business in the UDDI registry, then use the **Add a business** option on the **Advanced Publish** section of the Publish pane to add a new one.
2. Add a technical model:
   a. Select **Add a technical model** on the **Advanced Publish** section of the Publish pane.
   b. Enter the name as specified for the target namespace of your binding (or interface) WSDL file, then add a description (if required).
   c. Add a category of Type `unspsc` and value `wsdlSpec` (the Key name field can be left blank).
   d. Add an overview URL specifying the Web address for your binding WSDL file, then add a description (if required).

      Note: The binding and the service definition for your Web service might be held in separate WSDL files, therefore be careful to type the Web address of the WSDL file that defines the *binding*.
   e. Click **Publish Technical Model**.
3. Add a service:
   a. Select **Show owned entities** on the **Advanced Publish** section of the Publish pane.
   b. Select **Add a Service** for your business.
   c. Enter the name as specified for the target service in your WSDL file, then add a description (if required).

d. For the **Access point** verify that the correct Web address type is selected (for example `http` for an HTTP access point), then enter the value of the `soap:address` location (or its equivalent) from your service definition WSDL file (for example `http://yourhost:80/SimpleTest/servlet/rpcrouter`).
e. For the **Technical model** select **Add**, then find the required Technical model by entering a suitable prefix and selecting **Find technical models**, then enable the check box for the required Technical model and click **Update**.
f. Click **Publish Service**.

Results

The UDDI registry assigns a service key to your service, and publishes the service.

What to do next

After the service has been published you can get the service key from the target UDDI registry.

Here is an example of a full UDDI service key:

```
uddi:blade108node01cell:blade108node01:server1:default:6e3d106e-5394-44e3-be17-aca728ac1791
```

The service-specific part of this key is the final part:

```
6e3d106e-5394-44e3-be17-aca728ac1791
```

Creating a new WS-Security binding

Create a new WS-Security binding for use with service integration bus-enabled Web services. You use WS-Security bindings to secure the SOAP messages that pass between service requesters (clients) and inbound services, and between outbound services and target Web services.

Before you begin

Use this option to create WS-Security bindings that comply with either the Web Services Security (WS-Security) 1.0 specification, or the previous WS-Security specification, WS-Security Draft 13 (also known as the Web Services Security Core Specification).

Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

This topic assumes that you have got, from the owning parties, the WS-Security bindings for the client (in the case of an inbound service) and the target Web service (in the case of an outbound service).

You can only use WS-Security with Web service applications that comply with the *Web services for Java Platform, Enterprise Edition (Java EE)* or *Java Specification Requirements (JSR) 109* specification. For more information, see Web services security and Java Platform, Enterprise Edition security relationship. For information about how to make your Web service applications JSR-109 compliant, see Developing and deploying JAX-RPC Web services clients or Developing and deploying JAX-WS Web services clients .

About this task

WS-Security bindings provide the information that the run-time environment needs to implement the WS-Security configuration (for example “To sign the body, use this key”), You receive this security binding information direct from the service requester or target service provider, in the form of an `ibm-webservicesclient-bnd.xmi` file for the client, and an `ibm-webservices-bnd.xmi` file for the target Web service. You extract the information from these `.xmi` files, then manually enter it into the WS-Security bindings forms.

Bindings are administered independently from any Web service that uses them, so you can create a binding then apply it to many Web services.

WebSphere Application Server also includes a set of default WS-Security binding objects, as described in Default bindings and runtime properties for Web services security. However, if you are using either of the single server products WebSphere Application Server or WebSphere Application Server Express, then these default bindings are configured within the application server, and are not available for use with bus-enabled Web services.

Unlike most other configuration objects, when you create a WS-Security binding you can only define its basic aspects. To define the binding details you need to save the new binding, then reopen it for modification as described in Modifying an existing WS-Security binding.

To create a new WS-Security binding, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security bindings**. The WS-Security bindings collection form is displayed.
3. Click **New**. The New WS-Security binding wizard is displayed.
4. Use the wizard to assign the following general properties:
   a. Select the version of the WS-Security specification. Set this option to either Draft 13 (for a binding that complies with the WS-Security Draft 13 specification) or 1.0 (for a binding that complies with the Web Services Security (WS-Security) 1.0 specification.

      Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

   b. Specify the binding type.

      Set this option to one of the following binding types:

      For WS-Security Version 1.0:

      - *request consumer*, for use when consuming requests from a client to an inbound service.
      - *request generator*, for use when generating requests from an outbound service to a target Web service.
      - *response consumer*, for use when consuming responses from a target Web service to an outbound service.
      - *response generator*, for use when generating responses from an inbound service to a client.

      For WS-Security Draft 13:

      - *request receiver*, for use when receiving requests from a client to an inbound service.
      - *request sender*, for use when sending requests from an outbound service to a target Web service.
      - *response receiver*, for use when receiving responses from a target Web service to an outbound service.
      - *response sender*, for use when sending responses from an inbound service to a client.

   c. Specify the WS-Security binding.

      Give a name to this binding. This name must be unique and it must follow the following syntax rules:

      - It must not start with “.” (a period).
      - It must not start or end with a space.
      - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

(WS-Security 1.0 bindings only. Optional.) Select the **Use defaults** check box to create a convenient default binding for use in a development and test environment. If you select this option, the binding uses the WebSphere Application Server default set of binding information rather than any custom information that you might subsequently add. Note however that this default binding is by definition insecure, and is not for production use. You can also select or clear this check box when you modify an existing WS-Security binding.

Note: If you are creating a WS-Security 1.0 request generator binding, the Web address for the WS-Security 1.0 namespace is displayed in a selection list. This is the namespace used by WS-Security 1.0 to send a request, and you should not need to change this value. The other values included in the selection list refer to namespaces used by earlier versions of the WS-Security draft specification, and are included for backwards compatibility.

5. Click **Finish**. The general properties for this item are saved.

Results

If the processing completes successfully, the list of WS-Security bindings is updated to include the new binding. Otherwise, an error message is displayed.

What to do next

You are now ready to define the binding details as described in "Modifying an existing WS-Security binding."

Modifying an existing WS-Security binding

You can add or modify the configuration details for a WS-Security binding that is configured for use with service integration bus-enabled Web services. You use WS-Security bindings to secure the SOAP messages that pass between service requesters (clients) and inbound services, and between outbound services and target Web services.

About this task

WS-Security bindings provide the information that the run-time environment needs to implement the WS-Security configuration (for example "To sign the body, use this key"). You receive this security binding information direct from the service requester or target service provider, in the form of an `ibm-webservicesclient-bnd.xmi` file for the client, and an `ibm-webservices-bnd.xmi` file for the target Web service. You extract the information from these `.xmi` files, then manually enter it into the WS-Security bindings forms.

Bindings are administered independently from any Web service that uses them, so you can create a binding then apply it to many Web services.

To list the WS-Security bindings, and to view and modify their configuration details, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security bindings**. A list of WS-Security bindings is displayed in a WS-Security bindings collection form.

   Each available binding is flagged as one of the following binding types:

   For WS-Security Version 1.0:

   - *request consumer*, for use when consuming requests from a client to an inbound service.
   - *request generator*, for use when generating requests from an outbound service to a target Web service.
   - *response consumer*, for use when consuming responses from a target Web service to an outbound service.

- *response generator*, for use when generating responses from an inbound service to a client.

For WS-Security Draft 13:

- *request receiver*, for use when receiving requests from a client to an inbound service.
- *request sender*, for use when sending requests from an outbound service to a target Web service.
- *response receiver*, for use when receiving responses from a target Web service to an outbound service.
- *response sender*, for use when sending responses from an inbound service to a client.

Each available binding is also flagged as complying with either the Web Services Security (WS-Security) 1.0 specification or the WS-Security Draft 13 specification.

Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

3. Click the name of a WS-Security binding in the list. The current settings for this WS-Security binding are displayed.
4. Modify the configuration details for this WS-Security binding. For detailed reference information about each value that you can set, click on the associated link in the following tables:

WS-Security 1.0 request consumer	WS-Security 1.0 request generator
• Signing information • Encryption information • Token consumers • Key information • Key locators • Collection certificate store • Trust anchors • Properties	• Signing information • Encryption information • Token generators • Key information • Key locators • Collection certificate store • Properties

WS-Security 1.0 response generator	WS-Security 1.0 response consumer
• Signing information • Encryption information • Token generators • Key information • Key locators • Collection certificate store • Properties	• Signing information • Encryption information • Token consumers • Key information • Key locators • Collection certificate store • Trust anchors • Properties

WS-Security Draft 13 request receiver	WS-Security Draft 13 request sender
• Signing information • Encryption information • Trust anchors • Collection certificate store • Key locators • Trusted ID evaluators • Login mappings	• Signing information • Encryption information • Key locators • Login binding

WS-Security Draft 13 response sender	WS-Security Draft 13 response receiver
• Signing information • Encryption information • Key locators	• Signing information • Encryption information • Trust anchors • Collection certificate store • Key locators

5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of WS-Security bindings is redisplayed. Otherwise, an error message is displayed.

Deleting WS-Security bindings

Delete WS-Security bindings that are configured for use with service integration bus-deployed Web services.

About this task

When you remove a WS-Security binding that is currently used by one or more Web services on a service integration bus, the system removes the WS-Security binding for each associated Web service. To remove WS-Security bindings, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security bindings**. A list of WS-Security bindings is displayed in a WS-Security bindings collection form.
3. Select the check box for every WS-Security binding that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of WS-Security bindings is updated. Otherwise, an error message is displayed.

Creating a new WS-Security configuration

Create a new WS-Security configuration for use with service integration bus-deployed Web services. You use WS-Security configurations to secure the SOAP messages that pass between service requesters (clients) and inbound services, and between outbound services and target Web services.

Before you begin

Use this option to work with WS-Security configurations that comply with either the Web Services Security (WS-Security) 1.0 specification, or the previous WS-Security specification, WS-Security Draft 13 (also known as the Web Services Security Core Specification).

Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

This topic assumes that you have got, from the owning parties, the WS-Security configurations for the client (in the case of an inbound service) and the target Web service (in the case of an outbound service).

You can only use WS-Security with Web service applications that comply with the *Web services for Java Platform, Enterprise Edition (Java EE)* or *Java Specification Requirements (JSR) 109* specification. For more information, see Web services security and Java Platform, Enterprise Edition security relationship. For information about how to make your Web service applications JSR-109 compliant, see Developing and deploying JAX-RPC Web services clients or Developing and deploying JAX-WS Web services clients .

About this task

WS-Security configurations specify the level of security that you require (for example “The body must be signed”). This level of security is then implemented through the run-time information contained in a WS-Security binding. You receive the security configuration information direct from the service requester or target service provider, in the form of an `ibm-webservicesclient-ext.xmi` file for the client, and an `ibm-webservices-ext.xmi` file for the target Web service, which contain the information on the levels of security (integrity, confidentiality and identification) that are required. You extract the information from these `.xmi` files, then manually enter it into the WS-Security configuration forms.

Configurations are administered independently from any Web service that uses them, so you can create a configuration then apply it to many Web services. However, the security requirements for an inbound service (which acts as a target Web service) are significantly different to those required for an outbound service (which acts as a client). Consequently, configurations are further divided by service type (inbound or outbound).

Unlike most other configuration objects, when you create a WS-Security configuration you can only define its basic aspects. To define the details you save the new WS-Security configuration, then reopen it for modification as described in “Modifying an existing WS-Security configuration” on page 1050.

To create a new WS-Security configuration, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security configurations**. The WS-Security service configurations collection form is displayed.
3. Click **New**. The New WS-Security Service Configuration wizard is displayed.
4. Use the wizard to assign the following general properties:
   a. Select the version of the WS-Security specification. Set this option to either Draft 13 (for a configuration that complies with the WS-Security Draft 13 specification) or 1.0 (for a configuration that complies with the Web Services Security (WS-Security) 1.0 specification.

      Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.

   b. Specify the service type. If you are creating a configuration to secure the SOAP messages that pass between a service requester (client) and an inbound service (which acts as a target Web service), select Inbound Service. If you are creating a configuration to secure the SOAP messages that pass between an outbound service (which acts as a client) and a target Web service, select Outbound Service.
   c. Specify the WS-Security configuration type.

      Give a name to this configuration. This name must be unique across both WS-Security Version 1.0 and Draft 13 configurations, and it must follow the following syntax rules:

      - It must not start with “.” (a period).
      - It must not start or end with a space.
      - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

(Optionally) Specify an Actor URI for this configuration. WS-Security headers within the consumed request message are only processed if they have the specified Actor URI.

5. Click **Finish**. The general properties for this item are saved.

Results

If the processing completes successfully, the list of WS-Security configurations is updated to include the new configuration. Otherwise, an error message is displayed.

What to do next

You are now ready to define the configuration details as described in "Modifying an existing WS-Security configuration."

Modifying an existing WS-Security configuration

You can add or modify the configuration details for a WS-Security configuration that is configured for use with service integration bus-enabled Web services. You use WS-Security configurations to secure the SOAP messages that pass between service requesters (clients) and inbound services, and between outbound services and target Web services.

About this task

WS-Security configurations specify the level of security that you require (for example "The body must be signed"). This level of security is then implemented through the run-time information contained in a WS-Security binding. You receive the security configuration information direct from the service requester or target service provider, in the form of an `ibm-webservicesclient-ext.xmi` file for the client, and an `ibm-webservices-ext.xmi` file for the target Web service, which contain the information on the levels of security (integrity, confidentiality and identification) that are required. You extract the information from these `.xmi` files, then manually enter it into the WS-Security configuration forms.

Configurations are administered independently from any Web service that uses them, so you can create a configuration then apply it to many Web services. However, the security requirements for an inbound service (which acts as a target Web service) are significantly different to those required for an outbound service (which acts as a client). Consequently, configurations are further divided by service type (inbound or outbound).

To list the WS-Security configurations, and to view and modify their configuration details, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security configurations**. A list of WS-Security configurations is displayed in a WS-Security service configurations collection form.

   Each available configuration is flagged as either `Inbound` or `Outbound`. You use an inbound configuration to secure the SOAP messages that pass between a service requester (client) and an inbound service (which acts as a target Web service). You use an outbound configuration to secure the SOAP messages that pass between an outbound service (which acts as a client) and a target Web service.

   Each available configuration is also flagged as complying with either the Web Services Security (WS-Security) 1.0 specification or the WS-Security Draft 13 specification.

   Note: Use of WS-Security Draft 13 was deprecated in WebSphere Application Server Version 6.x. You should only use Draft 13 bindings to enable inter-operation between applications running in WebSphere Application Server Version 5.1 and Version 7.0, or to allow continued use of an existing Web services client application that is written to the WS-Security Draft 13 specification.
3. Click the name of a WS-Security configuration in the list. The current settings for this WS-Security configuration are displayed.

4. Modify the configuration details for this WS-Security configuration. For detailed reference information about each value that you can set, click on the associated link in the following table:

WS-Security 1.0 inbound configuration	WS-Security 1.0 outbound configuration
Request consumer • Required integrity – Message parts – Nonce – Time stamp • Required confidentiality – Message parts – Nonce – Time stamp • Required security token • Caller – Trust method - Properties – Properties • Add time stamp – Properties • Properties **Response generator** • Actor • Integrity – Message parts – Nonce – Time stamp • Confidentiality – Message parts – Nonce – Time stamp • Security Token • Add time stamp – Properties • Properties	**Request generator** • Actor • Integrity – Message parts – Nonce – Time stamp • Confidentiality – Message parts – Nonce – Time stamp • Security Token • Add time stamp – Properties • Properties **Response consumer** • Required integrity – Message parts – Nonce – Time stamp • Required confidentiality – Message parts – Nonce – Time stamp • Required security token • Caller – Trust method - Properties – Properties • Add time stamp – Properties • Properties

WS-Security Draft 13 inbound configuration	WS-Security Draft 13 outbound configuration
Request receiver • Required integrity • Required confidentiality • Login configuration – Custom authentication methods • ID assertion • Add received time stamp • Properties	**Request sender** • Actor • Integrity • Confidentiality • Login configuration • ID assertion • Add created time stamp • Properties
Response sender • Actor • Integrity • Confidentiality • Add created time stamp • Properties	**Response receiver** • Required integrity • Required confidentiality • Add received time stamp • Properties

5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of WS-Security configurations is redisplayed. Otherwise, an error message is displayed.

Deleting WS-Security configurations

Delete WS-Security configurations that are configured for use with service integration bus-enabled Web services.

About this task

When you remove a WS-Security configuration that is currently used by one or more Web services on a service integration bus, the system removes the WS-Security configuration for each associated Web service. To remove WS-Security configurations, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Web services** → **WS-Security configurations**. A list of WS-Security configurations is displayed in a WS-Security service configurations collection form.
3. Select the check box for every WS-Security configuration that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of WS-Security configurations is updated. Otherwise, an error message is displayed.

Passing SOAP messages with attachments through the service integration bus

The service integration technologies support Web services that use a SOAP binding to pass attachments in a MIME message.

Before you begin

See the restrictions detailed in Limitations in the support for SOAP with attachments.

About this task

The service integration bus supports SOAP messages that contain either old-style attachments (as described in the SOAP Messages with Attachments W3C Note) or attachments that use the Web Services-Interoperability (WS-I) Attachments Profile Version 1.0 (subject to the Limitations in the support for SOAP with attachments).

Attachments are carried through the service integration bus and passed to the inbound or outbound service. The content MIME type of each attachment is preserved. When the external target service for an outbound service is deployed to a Java API for XML-based Remote Procedure Call (JAX-RPC) compliant server, you can access the attachments on the target service using the `javax.activation.DataHandler` handler.

If the bus receives a message that contains attachments, and the bus subsequently rewrites the message, then the generated message uses the same attachment style as the received message. If you need to transform attachments from one style to another, you can use a mediation to modify the message.

For more information, see the following topics:

- "SOAP Messages with Attachments: WSDL examples" on page 1054
- "Supporting bound attachments: WSDL examples" on page 1054
- "Locating an attachment using swaref"
- Writing a mediation that maps between attachment encoding styles

Locating an attachment using swaref

About this task

When a Web Services-Interoperability (WS-I) Attachments Profile Version 1.0 message uses a SOAP with attachments reference (swaref) to refer to an attachment, the swaref might refer to either bound or non-bound attachments, and the swaref might refer to a single attachment several times. To enable you to locate the correct attachment, service integration technologies stores the value of the URI that is encoded in the message within the SDO data graph for the message body.

When storing the value of an element (or attribute) of type swaref in the data graph, service integration technologies stores the complete URI from the message instance. Therefore when you retrieve the URI you need to remove `cid:` from the beginning of the retrieved value to find the Content ID of the referenced attachment.

Example

The following example shows how to use the value of a swaref element to locate the correct attachment. This example uses the RPC/Literal WSDL and SOAP message from section 4.4 of Web Services-Interoperability (WS-I) Attachments Profile Version 1.0:

```
DataObject infoNode = graph.getRootObject().getDataObject("info");
String contentId = infoNode.getString("body/ClaimDetail/ClaimForm");

// Cut off the "cid:" part of the string
contentId = contentId.substring(4);

// Locate the value of the attachment
DataObject attachmentEntry =
    infoNode.getDataObject("attachments[contentId=" + contentId + "]");
byte[] data = attachmentEntry.getBytes("data");
```

SOAP Messages with Attachments: WSDL examples
Example

The following example WSDL illustrates a simple operation that has one attachment called `attch`:

```
<binding name="MyBinding" type="tns:abc" >
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="MyOperation">
    <soap:operation soapAction=""/>
    <input>
      <mime:multipartRelated>
        <mime:part>
          <soap:body parts="part1 part2 ..." use="encoded" namespace="http://mynamespace"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
        </mime:part>
        <mime:part>
          <mime:content part="attch" type="text/html"/>
        </mime:part>
      </mime:multipartRelated>
    </input>
  </operation>
</binding>
```

In this type of WSDL extension:

- There must be a `part` attribute (in this example `attch`) on the input message for the operation (in this example `MyOperation`). There can be other input parts to `MyOperation` that are not attachments.
- In the binding input there must either be a `<soap:body>` tag or a `<mime:multipartRelated>` tag, but not both.
- For MIME messages, the `<soap:body>` tag is inside a `<mime:part>` tag. There must only be one `<mime:part>` tag that contains a `<soap:body>` tag in the binding input and that must not contain a `<mime:content>` tag as well, because a content type of `text/xml` is assumed for the `<soap:body>` tag.
- There can be multiple attachments in a MIME message, each described by a `<mime:part>` tag.
- Each `<mime:part>` tag that does not contain a `<soap:body>` tag contains a `<mime:content>` tag that describes the attachment itself. The `type` attribute inside the `<mime:content>` tag is not checked or used by the service integration bus. It is there to suggest to the application using the service integration bus what the attachment contains. Multiple `<mime:content>` tags inside a single `<mime:part>` tag means that the back end service expects a single attachment with a type specified by one of the `<mime:content>` tags inside that `<mime:part>` tag.
- The `parts="..."` attribute inside the `<soap:body>` tag is assumed to contain the names of all the SOAP parts in the message, but not the attachment parts. If there are only attachment parts, specify `parts=""` (empty string). If you omit the `parts` attribute altogether, then the service integration bus assumes ALL parts including the attachments - which causes the attachments to appear twice.

In your WSDL you might have defined a schema for the attachment (for instance as a `binary[]`). The service integration technologies silently ignore this mapping and treat the attachment as a Data Handler.

You do not need to mention unreferenced attachments in the WSDL bindings.

Supporting bound attachments: WSDL examples
About this task

Web Services-Interoperability (WS-I) Attachments Profile Version 1.0 defines a convention for constructing the Content ID for a bound attachment. This convention encodes the message part name. Consequently, service integration technologies can recognize a bound attachment whether or not the SOAP body contains elements representing that message part. The convention for constructing a Content ID is as follows:

name=uuid@domain

where *name* is the name of the message part that is being encoded, *uuid* is a globally unique identifier, and *domain* is a domain identifier (for example `my.example.com`).

Note: This approach differs from the SOAP Messages with Attachments encoding scheme, which does not define a conventions for the Content ID but does use elements within the SOAP body to indicate that the message part is encoded as an attachment.

In order to distinguish between the cases, service integration technologies assumes that if a message attachment follows the Version 1.0 convention for constructing the Content ID, then it is a Version 1.0 message.

Example

The following WSDL fragment is for a bound attachment, with message instances that follow both styles:

```
<wsdl:binding name="BoundSoapBinding" type="intf:BoundPortType">
    <soap:binding style="rpc"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="bound">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <mime:multipartRelated>
          <mime:part>
            <soap:body parts="stringIn" namespace="http://bound"
                       use="literal"/>
          </mime:part>
          <mime:part>
            <mime:content part="attachIn" type="text/xml"/>
          </mime:part>
        </mime:multipartRelated>
      </wsdl:input>
```

The following WSDL fragment is for a SOAP instance using Version 1.0 encoding. In this fragment, the message body contains no mention of the `attachIn` part, and the Content ID of the attachment identifies the part that is being encoded.

```
--myBoundary
Content-Type: text/xml
Content-Transfer-Encoding: 7bit
Content-Id: <myStartID>

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns0:bound xmlns:ns0="http://bound">
      <stringIn>some string data</stringIn>
    </ns0:bound>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--myBoundary
Content-Type: text/xml
Content-Transfer-Encoding: 7bit
Content-Id: <attachIn=someUUID@some.domain.name>

<someOtherXMLElement/>
--myBoundary--
```

The following WSDL fragment is for a SOAP instance using SOAP Messages with Attachments encoding. In this fragment, the message body does contain a reference to the bound attachment, and the Content ID of the attachment is not constrained.

```
--myBoundary
Content-Type: text/xml
Content-Transfer-Encoding: 7bit
```

```
Content-Id: <myStartID>

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns0:bound xmlns:ns0="http://bound">
      <stringIn>some string data</stringIn>
      <attachIn href="cid:notTheStart"/>
    </ns0:bound>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--myBoundary
Content-Type: text/xml
Content-Transfer-Encoding: 7bit
Content-Id: <notTheStart>

<someOtherXMLElement/>
--myBoundary--
```

In the previous two cases there is sufficient information in the message to identify the bound attachment, and in both cases service integration technologies places a bound attachment entry in the attachments list, and places the data from the attachment into the body section of the data graph.

Using WS-Notification for publish and subscribe messaging for Web services

WS-Notification enables Web services to use the “publish and subscribe” messaging pattern.

About this task

You use publish and subscribe messaging to publish one message to many subscribers. In this pattern a producing application inserts (publishes) a message (event notification) into the messaging system having marked it with a topic that indicates the subject area of the message. Consuming applications that have subscribed to the topic in question, and have appropriate authority, all receive an independent copy of the message that was published by the producing application.

Through the implementation of WS-Notification in WebSphere Application Server, you can achieve the following goals:

- Leverage existing service integration technologies and Web services components to deliver WS-Notification functionality.
- Interoperate with other publish and subscribe messaging clients (for example Java Message Service (JMS), WebSphere MQ) and with alternative message brokering products.

For more information about using WS-Notification, see the following topics:

- “Learning about WS-Notification” on page 1059
- “Accomplishing common WS-Notification tasks” on page 1057
- “Configuring WS-Notification resources” on page 1079
- “Securing WS-Notification” on page 1066
- Developing applications that use WS-Notification

What to do next

Note:

- For development use on a single server only, you can use a script to configure the necessary resources to get up and running quickly with WS-Notification.

- In WebSphere Application Server Version 6.1, support for WS-Notification is based on a pre-final approval public review draft of the WS-Notification standards. In Version 7.0, this support is extended to cover the final approved standards. The differences between the two versions of the standards are small, and your existing WS-Notification services and client applications will continue to work unchanged. However, you might choose to upgrade your existing applications as described in Enabling WebSphere Application Server Version 6.1 client applications to handle the additional error conditions introduced in the final Version 1.3 WS-Notification standards.

Accomplishing common WS-Notification tasks

The specific steps that you need to perform to implement the most common WS-Notification tasks. For some tasks the steps are subdivided by role, and each sub-task gives the steps that are completed by either the solution architect, the system administrator or the application developer.

About this task

In the following list the sub-tasks are grouped by role. For more information about the roles, see "WS-Notification roles and goals."

- "Solution architect" on page 1058 sub-tasks:
  - "Learning about WS-Notification" on page 1059.
  - Selecting a hardware and software product combination for the enterprise that supports the WS-Notification standards.
  - Designing a server topology to host the applications, in accordance with particular WS-Notification topologies.
- "System administrator" on page 1058 sub-tasks:
  - "Using a script to get up and running quickly with WS-Notification" on page 1059.
  - "Configuring a WS-Notification service for use only by WS-Notification applications" on page 1063.
  - "Providing access for WS-Notification applications to an existing bus topic space" on page 1065.
  - "Securing WS-Notification" on page 1066.
  - "Configuring JAX-WS handlers" on page 1068.
  - "Applying a JAX-WS handler list to a WS-Notification service" on page 1068.
  - "Configuring a Version 7.0 WS-Notification service with Web service QoS" on page 1070.
  - "Configuring WS-Notification for reliable notification" on page 1071.
  - "Migrating a Version 6.1 WS-Notification configuration from WebSphere Application Server Version 6.1 to Version 7.0" on page 1073.
  - "Preparing a migrated Version 6.1 WS-Notification configuration for reliable notification" on page 1073.
  - "Interacting at run time with WS-Notification" on page 1077.
  - "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.
- "Application developer" on page 1059 sub-tasks:
  - Developing applications that use WS-Notification
  - Writing a WS-Notification application that exposes a Web service endpoint.
  - Writing a WS-Notification application that does not expose a Web service endpoint.
  - Filtering the message content of publications.

WS-Notification roles and goals

This topics lists a set of computing roles that members of your organization might perform, and explains how you can use WS-Notification to help meet the goals of each role.

For a general description of each of the following roles, see WebSphere Application Server roles and goals.

Enterprise architect

IT environments are currently evolving towards the following concepts:

- Service Oriented Architecture (SOA)
- Enterprise Service Bus (ESB)

The goal of the enterprise architect might be to guide their organization towards appropriate utilization of these concepts to maximize the efficiency and responsiveness of the business as a whole.

WS-Notification enables publish and subscribe communication patterns (such as a stock ticker) to be exposed using Web services in an SOA environment. This is done through open standards, enabling straightforward replacement of the service implementation. It promotes easy exchange of data between suppliers and customers through use of standard Web service operations and prevents vendor lock-in or adoption of proprietary standards.

WebSphere Application Server also allows WS-Notification to be used as an on- or off-ramp to an ESB, providing seamless interchange of data between different types of client connected to the bus.

Solution architect

The main goal of the solution architect is to design a solution that supports the specification set by the enterprise architect. This might include providing an environment in which Web service applications can participate in publish and subscribe messaging patterns. This participation might also include the requirement to be able to exchange event notifications between Web service clients and other clients of the enterprise service bus.

To create a design, the solution architect completes the following broad steps:

- Learn about the support provided for WS-Notification in WebSphere Application Server.
- Select a hardware and software product combination for the enterprise that supports the WS-Notification standards.
- Design a server topology to host the applications, in accordance with the particular WS-Notification topologies that are to be implemented.

System administrator

For the specific steps that the system administrator performs to help implement common WS-Notification tasks, see the following topics:

- “Using a script to get up and running quickly with WS-Notification” on page 1059.
- “Configuring a WS-Notification service for use only by WS-Notification applications” on page 1063.
- “Providing access for WS-Notification applications to an existing bus topic space” on page 1065.
- “Securing WS-Notification” on page 1066.
- “Configuring JAX-WS handlers” on page 1068.
- “Applying a JAX-WS handler list to a WS-Notification service” on page 1068.
- “Configuring a Version 7.0 WS-Notification service with Web service QoS” on page 1070.
- “Configuring WS-Notification for reliable notification” on page 1071.
- “Migrating a Version 6.1 WS-Notification configuration from WebSphere Application Server Version 6.1 to Version 7.0” on page 1073.
- “Preparing a migrated Version 6.1 WS-Notification configuration for reliable notification” on page 1073.
- “Interacting at run time with WS-Notification” on page 1077.
- “Publishing the WSDL files for a WS-Notification application to a ZIP file” on page 1079.

Application developer

If the solution architect specifies a requirement to insert event notifications into the system (that is publish messages) or receive event notifications from the system as a result of creating a subscription containing an interest profile, then the application developer can use WS-Notification to meet this requirement.

There are various patterns of producing and consuming application defined by WS-Notification that are available for use by the application developer, depending upon the exact requirements of the application in question. These options are explored in the following common WS-Notification tasks:

- Writing a WS-Notification application that exposes a Web service endpoint.
- Writing a WS-Notification application that does not expose a Web service endpoint.

See also Developing applications that use WS-Notification and Filtering the message content of publications.

Learning about WS-Notification

WS-Notification enables Web services to use the “publish and subscribe” messaging pattern. Use these topics to learn more about WS-Notification.

About this task

You use publish and subscribe messaging to publish one message to many subscribers. In this pattern a producing application inserts (publishes) a message (event notification) into the messaging system having marked it with a topic that indicates the subject area of the message. Consuming applications that have subscribed to the topic in question, and have appropriate authority, all receive an independent copy of the message that was published by the producing application.

To learn about the WS-Notification implementation in WebSphere Application Server, see the following topics:

- WS-Notification: Overview
- WS-Notification: Benefits
- WS-Notification and end-to-end reliability
- WS-Notification terminology
- WS-Notification: How client applications interact at runtime
- WS-Notification: Supported bindings
- WS-Notification and policy set configuration
- Reasons to create multiple WS-Notification services in a bus
- Reasons to create multiple WS-Notification service points
- Options for associating a permanent topic namespace with a bus topic space
- WS-Notification topologies

Using a script to get up and running quickly with WS-Notification

Use a jython script to configure the necessary resources to get up and running quickly with WS-Notification in WebSphere Application Server.

Before you begin

The example script provided in this topic is intended for development use on a single server only, and not for use in production or network deployment environments.

About this task

You can use the example script to configure a default set of resources that enable you to connect WS-Notification applications for development purposes. When executed, the script takes the following actions:

1. It searches in the configuration for an existing service integration bus, and creates one if necessary.
2. It searches for an existing bus member, and if one is not found it adds the (standalone) server to the bus, using the default data source.
3. It searches for an existing service integration bus topic space destination, and creates one if necessary.
4. It creates a Version 6.1 or Version 7.0 WS-Notification service.
5. It creates a Version 6.1 or Version 7.0 WS-Notification service point on the local server for a SOAP over HTTP binding.
6. It creates a WS-Notification permanent topic namespace to reference the service integration bus topic space that was found or created in step 3.
7. It saves the configuration, and describes where to find the WSDL for the new notification broker Web service that has been exposed.

To use the example script, complete the following steps:

1. Save the script to the file system using a name of your choice (for example `wsnQuickStart.py`).
2. Modify the *hostRoot* variable defined at the top of the script to point to the HTTP port of the local server (usually 9080).
3. Install and configure the SDO repository.
4. Start the server, then execute the following command. If you saved the script with a name other than `wsnQuickStart.py`, then use that name instead.

   ```
   wsadmin -f wsnQuickStart.py
   ```

Example

Here is the example script:

```
#############################################################################
# WS-Notification QuickStart script                                         #
#                                                                           #
# This Jython script will quickly create the basic resources required in order to     #
# start using WS-Notification in WebSphere Application Server Version 6.1 or later    #
#                                                                           #
# Before executing it you must modify the variables defined below to match your       #
# configuration settings.                                                   #
#                                                                           #
# Note:                                                                     #
#    - This script is not intended for production use, and is intended for use on     #
#      a standalone server (not network deployment) only.                   #
#    - The script will search the configuration for an existing bus, and if one is    #
#      not found then a new bus will be created                             #
#    - It will then look for an existing Bus Member on the chosen bus. If one is not  #
#      found then one will be created using the default File Store           #
#    - It will then look for an existing service integration bus topic space. If one  #
#      is not found then it will create one.                                #
#                                                                           #
# Execute the script by typing;                                             #
#   wsadmin -f wsnQuickStart.py                                             #
#                                                                           #
#############################################################################

#########################################################
# Configuration variables                               #
```

```
#                                                           #
# Set the following variables to match your configuration #
#############################################################
# The URL root of HTTP port on the local server
hostRoot = "http://xyz.ibm.com:9080"
# The type of WS-Notification service you want to create (Version 6.1 or Version 7.0)
wsnServiceType = "V7.0"

#############################################################################################
# Now create the configuration objects using the variables defined above              #
#############################################################################################

# These variables are arbitrary choices and need not be set unless desired.
wsnServiceName = "myWSNService"+wsnServiceType
wsnServicePointName = "myWSNServicePoint"+wsnServiceType
eplName = "myNewEPL"
tnsNamespaceURI = "http://example.org/topicNamespace/example1"

# General environment variables
nodeName = AdminTask.listNodes().split("\n")[0].rstrip()
server = AdminTask.listServers().split("\n")[0].rstrip()

print "#############################################################"
print "# Check the pre-requisites before we begin                #"
print "#############################################################"
# Check for the existence of the bus
requiresRestart = "false"
myBuses = AdminTask.listSIBuses().split("\n")
myBus = myBuses[0].rstrip()

if (myBus == ""):
        print "  *** Creating new bus "
        myBus = AdminTask.createSIBus("-bus MySampleBus -busSecurity false
                                       -scriptCompatibility 6.1")
        requiresRestart = "true"
#endIf
siBusName = AdminConfig.showAttribute(myBus, "name" )
print "  service integration bus name: "+siBusName+" "

# Check for the existence of the bus member
busMembers = AdminTask.listSIBusMembers(" -bus "+siBusName).split("\n")
myBusMember = busMembers[0].rstrip()

if (myBusMember == ""):
        print ""
        print "  *** Creating new Bus Member "
        busMemberName = AdminConfig.showAttribute(server, "name")
        myBusMember = AdminTask.addSIBusMember(" -bus "+siBusName+"
                        -node "+nodeName+" -server "+busMemberName)
        print ""
        requiresRestart = "true"
else:
        nodeName = AdminConfig.showAttribute(myBusMember, "node")
        busMemberName = AdminConfig.showAttribute(myBusMember, "server")
#endElse
print "  service integration bus Member on node: "+nodeName+" "
print "                                 on server: "+busMemberName+" "

# Find a topic space to use
topicSpaces = AdminTask.listSIBDestinations(" -bus "+siBusName+"
                                  -type TopicSpace ").split("\n")
tSpace = topicSpaces[0].rstrip()
```

```
if (tSpace == ""):
        print "  *** Creating a Topic Space "
        tSpace = AdminTask.createSIBDestination(" -bus "+siBusName+"
                        -node "+nodeName+" -server "+myBusMember+"
                        -name MyTopicSpace -type TopicSpace")
        print ""

#endIf
siBusTopicSpaceName = AdminConfig.showAttribute(tSpace, "identifier" )
print "     service integration bus topic space: "+siBusTopicSpaceName+" "

print ""
print "############################################################"
print "# Create the WS-Notification service                       #"
print "############################################################"
newService = AdminTask.listWSNServices(" -name "+wsnServiceName+"
                        -bus "+siBusName).split("\n")[0].rstrip()

if (newService == ""):
        newService = AdminTask.createWSNService(" -name "+wsnServiceName+"
                        -bus "+siBusName+" "+" -type "+wsnServiceType)
        print "WS-Notification service created: "+wsnServiceName+" "
        print "                          on bus: "+siBusName+" "
else:
        print "WS-Notification service '"+wsnServiceName+"' "
        print "  already exists on bus '"+siBusName+"' "
#endElse

print ""
print "############################################################"
print "# Create the WS-Notification service point                 #"
print "############################################################"
eplURLRoot = hostRoot+"/wsn"
wsdlURLRoot = hostRoot+"/SIBWS/wsdl"

newServicePoint = AdminTask.listWSNServicePoints(newService, "
               -name "+wsnServicePointName+" " ).split("\n")[0].rstrip()

if (newServicePoint == ""):
        if (wsnServiceType == "V7.0"):
            newServicePoint = AdminTask.createWSNServicePoint(newService, "
            -name "+wsnServicePointName+" -node "+nodeName+" -server "+busMemberName)
        else:
            newServicePoint = AdminTask.createWSNServicePoint(newService, "
            -name "+wsnServicePointName+"
            -node "+nodeName+" -server "+busMemberName+" -eplName "+eplName+"
            -eplURLRoot "+eplURLRoot+" -eplWSDLServingURLRoot "+wsdlURLRoot+" " )
        print "WS-Notification service point created: "+wsnServicePointName+" "
        print "                         on bus member: "+busMemberName+" "
        print "                                  on node: "+nodeName+" "
else:
        print "WS-Notification service point '"+wsnServicePointName+"' "
        print "already exists on WS-Notification service '"+wsnServiceName+"' "
#endElse

print ""
print "############################################################"
print "# Create the WS-Notification permanent topic namespace    #"
print "############################################################"
newTopicNamespace = AdminTask.listWSNTopicNamespaces(newService, "
                    -namespace "+tnsNamespaceURI+" ").split("\n")[0].rstrip()

if (newTopicNamespace == ""):
```

```
    newTopicNamespace = AdminTask.createWSNTopicNamespace(newService, "
      -namespace "+tnsNamespaceURI+" -busTopicSpace "+siBusTopicSpaceName+"
      -reliability RELIABLE_PERSISTENT")
  print "WS-Notification topic namespace created: "+tnsNamespaceURI+" "
  print "                         bus topic space: "+siBusTopicSpaceName+" "
else:
  print "WS-Notification permanent topic namespace already exists: "
        +tnsNamespaceURI+" "
#endElse

################################################################################
# All the objects have been created - inform the user where to proceed next    #
################################################################################

print ""
print "############################################################"
print "# Summary                                                  #"
print "############################################################"

# Calculate where you would find the WSDL for the new service.
if (wsnServiceType == "V7.0"):
  print "IMPORTANT: Because you've created a Version 7.0 service"
  print "you need to start the newly installed application;"
  print "           WSN_"+wsnServiceName+"_"+wsnServicePointName
  print ""
  wsdlLocation = hostRoot+"/"+wsnServiceName+wsnServicePointName
                       +"NB/NotificationBroker?wsdl"
else:
  print "IMPORTANT: Because you've created a Version 6.1 service"
  print "you need to start 2 newly installed applications;"
  serverName = AdminConfig.showAttribute(server, "name")
  print "           "+eplName+"."+nodeName+"."+serverName
  print "           sibws."+nodeName+"."+serverName
  print ""
  wsdlLocation = hostRoot+"/wsn/soaphttpengine/"+siBusName+"/"
                 +wsnServiceName+"NotificationBroker/"
                 +wsnServicePointName+"NotificationBrokerPort?wsdl"
print " The WSDL for the new service can be viewed at the following location; "
print "   "+wsdlLocation+" "
print ""
print " Your web service applications can publish and subscribe to any topics in the namespace; "
print "   "+tnsNamespaceURI+" "
print ""
if (requiresRestart == "true"):
  print " You must now restart the server for the changes to take effect. "
  print ""
#endIf

print ""
print "############################################################"
print "# Save the configuration and exit                          #"
print "############################################################"
AdminConfig.save()
sys.exit()
```

Configuring a WS-Notification service for use only by WS-Notification applications

Define all the necessary objects to configure a WS-Notification service and associated service points.

Before you begin

This task assumes that you have already configured one or more application servers, and that there are no other messaging resources configured..

For Version 6.1 WS-Notification services, you must also create an endpoint listener configuration on each application server that you want to use to host a WS-Notification service.

About this task

You can configure one or more servers, each hosting a WS-Notification service to which Web services applications can connect, as described in the Simple Web services topology.

1. Use the information given in “Creating a bus” on page 1139 to create a service integration bus and add your existing server or servers as bus members. When you add a bus member, a messaging engine is created in the server. You choose the database that hosts the persistent storage required by the messaging engine. You can either use the default JDBC data source and Apache Derby JDBC provider, or you can configure a data source for use with your preferred database product.
2. Using the administrative console, navigate to **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form for this bus is displayed.
3. Click **New**. The “New WS-Notification service” wizard is displayed.
4. Complete steps 1 to 4 of the wizard, as described in “Creating a new Version 6.1 WS-Notification service” on page 1089 or “Creating a new Version 7.0 WS-Notification service” on page 1081. Pay particular attention to the following settings:
   - In wizard step 1: “Configure name, description, service integration bus and dynamic topic namespace settings”, you select the service integration bus to use to host the messaging resources. Choose the bus that you created at the beginning of this task.
   - In wizard step 4: “Create WS-Notification service points”, you select the bus member that is to host the new service. On a single server system there is only one option here. For Version 6.1 WS-Notification services, you also select the endpoint listener application to use to expose the service. Choose the endpoint listener application that you configured before you began this task.
5. Optional: In wizard step 5: “Create permanent topic namespaces”, configure the WS-Notification topic namespace to provide access to a service integration bus topic space:
   - Enter the topic namespace URI that you want WS-Notification applications to use when referring to the service integration bus topic space. This must be unique within the WS-Notification service, and is conventionally a URI related to your organization. For example `http://www.myorganization.com`.
   - Because there are no existing service integration bus topic spaces in your bus the “Use an existing service integration bus topic space” radio button and associated selection list are unavailable. To create a new bus topic space, enter your chosen name.
6. Complete wizard step 6: “Summary”.

   Check that the summary of the actions taken by the wizard is as you expected, then click **Finish**. If the processing completes successfully, the list of WS-Notification services is updated to include the new WS-Notification service. Otherwise, an error message is displayed.
7. Save your changes to the master configuration. You do not need to restart the server for these changes to take effect. However, you do need to start the endpoint listener or enterprise application associated with the service point that was created in step 4 of the wizard.

What to do next

For Version 7.0 WS-Notification services, you can view the URL to which WS-Notification applications connect by looking in the NotificationBroker.WSDL file for the NotificationBroker application. To view this file, see “Publishing the WSDL files for a WS-Notification application to a ZIP file” on page 1079.

For Version 6.1 WS-Notification services, you can view the URL to which WS-Notification applications connect by navigating to **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **NotificationBroker inbound port settings**.

To extend this configuration so that Web service applications can interact with non-Web service applications, see “Providing access for WS-Notification applications to an existing bus topic space.”

Providing access for WS-Notification applications to an existing bus topic space

Configure a solution such that Web services clients can share event notifications with other clients of a service integration bus.

Before you begin

This task assumes that you have an existing service integration bus, configured with at least one bus member and a bus topic space. For more information, see “Creating a bus” on page 1139.

About this task

This task focuses on the “Create permanent topic namespaces” step that is part of the “New WS-Notification service” wizard.

You can configure WS-Notification so that Web service applications receive event notifications generated by other clients of the service integration bus such as JMS clients. Similarly, Web service applications can generate notifications to be received by other client types. This configuration is described in the Topology for WS-Notification as an entry or exit point to the service integration bus. You achieve this configuration by creating a permanent topic namespace that allows messages to be shared between Web service and non-Web service clients of the bus. Specifically, you create a permanent topic namespace that links the service integration bus topic space used by messaging clients to a WS-Notification topic namespace URI.

For more information about programming the client applications, including example code for cross-streaming from a JMS client, see Sharing event notification messages with other bus client applications.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
3. Click **New**. The “New WS-Notification service” wizard is displayed.
4. Complete steps 1 to 4 of the wizard, as described in “Creating a new Version 6.1 WS-Notification service” on page 1089 or “Creating a new Version 7.0 WS-Notification service” on page 1081.
5. In wizard step 5:“ Create permanent topic namespaces”, configure the WS-Notification topic namespace to provide access to the existing service integration bus topic space:
   - Enter the topic namespace URI that you want WS-Notification applications to use when referring to the service integration bus topic space. This must be unique within the WS-Notification service, and is conventionally a URI related to your organization. For example `http://www.myorganization.com`.
   - Enable the “Use an existing service integration bus topic space” radio button, then choose the name of your chosen service integration bus topic space from the selection list.
6. Complete wizard step 6: “Summary”.

   Check that the summary of the actions taken by the wizard is as you expected, then click **Finish**. If the processing completes successfully, the list of WS-Notification services is updated to include the new WS-Notification service. Otherwise, an error message is displayed.
7. Save your changes to the master configuration. You do not need to restart the server for these changes to take effect. However, you do need to start the endpoint listener or enterprise application associated with the service point that was created in step 4 of the wizard.

Results

WS-Notification applications can now connect to the WS-Notification service point and insert or receive event notifications from the service integration bus topic space.

What to do next

For Version 7.0 WS-Notification services, you can view the URL to which WS-Notification applications connect by looking in the NotificationBroker.WSDL file for the NotificationBroker application. To view this file, see “Publishing the WSDL files for a WS-Notification application to a ZIP file” on page 1079.

For Version 6.1 WS-Notification services, you can view the URL to which WS-Notification applications connect by navigating to **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **NotificationBroker inbound port settings**.

Securing WS-Notification

The WS-Notification security implementation requires that a user identity is flowed in requests for WS-Notification services. This identity is used to authenticate the client application and check that the client is authorized to invoke the requested operation, and to access the underlying service integration bus topic spaces and topic resources.

About this task

WS-Notification uses the same mechanisms as other Web services to provide an authenticated identity. For example WS-Security or HTTP Basic Authentication.

There are three parts to configuring secure access to WS-Notification:

- Securing the communication channel between the application and the server.
- Authorizing the application to invoke the NotificationBroker.
- Authorizing the application to access the resources of the service integration bus.

If messaging security is enabled, and the WS-Security or HTTP Basic Authentication components are not configured to flow a user identity in WS-Notification requests, then all such requests are treated as unauthenticated and can only access messaging resources that are accessible by the WebSphere Application Server “everyone” group.

1. Secure the communication between the application and the server:
   a. Provide security for inbound requests and associated responses by configuring the WS-Notification service point.
      - For Version 7.0 WS-Notification service points, attach security-enabled policy sets to the application associated with the service point. For Version 6.1 WS-Notification service points, configure security for the inbound ports associated with the service point.
      - If you are using SOAP over HTTP as the binding for your WS-Notification service point, modify it to use SOAP over HTTPS as described in Configuring secure access to WS-Notification service points using SOAP over HTTPS.
      - If you are using SOAP over JMS as the binding (for Version 6.1 WS-Notification services), configure the JMS connection factory used by the client application to use a secure communication protocol to communicate with the JMS provider. Exactly how you do this depends upon the JMS provider. If you are using the service integration bus as the JMS provider, configure the client to use SSL to communicate with the server by setting the **target inbound transport chain** to `InboundSecureMessaging` as described in How JMS applications connect to a messaging engine on a bus and its related tasks.
   b. Provide security for outbound requests (for example notifications from the server to subscribed consumers) by configuring the WS-Notification service.

- For Version 7.0 WS-Notification services, the steps involved are similar to those for applying security to JAX-WS Web service clients except that any binding or configuration created is applied to the WS-Notification service. For more information, see Securing JAX-WS Web services using message level security.
- For Version 6.1 WS-Notification services, the steps involved are similar to those for applying security to service integration bus-enabled Web services outbound ports except that any binding or configuration created is applied to the WS-Notification service. For more information, see Securing bus-enabled Web services and its sub-topics, notably Invoking outbound services over HTTPS.

c. You can also use WS-Security to sign or encrypt SOAP messages.
- For Version 7.0 WS-Notification services, see Signing and encrypting message parts using policy sets.
- For Version 6.1 WS-Notification services, see Configuring secure transmission of SOAP messages using WS-Security.

2. Authorize the application to invoke the NotificationBroker:

a. Configure the client application to provide an appropriate identity.

To authorize a Web service application to communicate with the server, the application must identify itself as running as a particular authenticated identity. The mechanism for doing this depends upon the type of Web service binding you are using:
- If you are using SOAP over HTTP Web service bindings, use either HTTP Basic Authentication or WS-Security to provide the authenticated identity.
- If you are using SOAP over JMS Web service bindings (for Version 6.1 WS-Notification services), use WS-Security to provide an authenticated identity.

b. Configure the server to authorize the client application identity to carry out the required operations.
- For Version 7.0 WS-Notification services, you can use Web services policy sets such as the “Username WS-I RSP default” or “Username WSSecurity default” policy sets to apply authorization to the Web services that are deployed in the enterprise application associated with a service point.
- For Version 6.1 WS-Notification services, you can apply authorization to the whole of an inbound service (for example the NotificationBroker endpoint of a WS-Notification service point) as described in Password-protecting inbound services, or configure authorization constraints independently for each Web service operation as described in Password-protecting a Web service operation.

3. Authorize the application to access the resources of the service integration bus.

Service integration bus security uses role-based authorization. When a user is assigned to a role, the user is granted all of the permissions that the role contains. By administering authorization permissions, you can control user access to a bus and to its resources when messaging security is enabled.

a. Authorize the application identity to be able to connect to the service integration bus, as described in “Administering the bus connector role” on page 1282.

b. When the application can connect to the bus, grant the application access to the appropriate destinations on the bus.

You can determine which service integration bus topic spaces are required, by checking which WS-Notification topic namespaces are used by the application then looking at the appropriate WS-Notification permanent topic namespace to find the service integration bus topic space to which it maps. You can then grant authorization (for example the Sender or Receiver roles) for the authenticated identity to access that topic space as described in “Administering destination roles” on page 1241.

c. After the client application has been authorized to access the appropriate topic space destination, you might also need to authorize the client application to access the individual topics within the topic space destination as described in “Administering topic roles” on page 1302.

For general information about configuring access to the service integration bus, see bus security.

Configuring JAX-WS handlers

A JAX-WS handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. You can create JAX-WS handlers, chain them together in the form of a handler list, then apply the handler list to a Version 7.0 WS-Notification service point (for inbound invocation handling) or WS-Notification service (for outbound invocation handling).

About this task

The Java API for XML-based Web services (JAX-WS) provides you with a standard way of developing interoperable and portable Web services. To create a JAX-WS handler, you can use a tool such as IBM Rational Application Developer. To enable handlers to perform more complex operations, you chain them together into handler lists. You associate each handler list with one or more Version 7.0 WS-Notification services or service points, so that the handler list can monitor WS-Notification activity and take appropriate action depending upon the sender and content of each inbound or outbound message.

Detailed instructions on how to configure JAX-WS handlers and handler lists for use with Version 7.0 WS-Notification services are provided in the following topics:

- Load JAX-WS handler classes.
- Create a new JAX-WS handler configuration.
- Modify an existing JAX-WS handler configuration.
- Delete JAX-WS handler configurations.
- Create a new JAX-WS handler list.
- Modify an existing JAX-WS handler list.
- Delete JAX-WS handler lists.

Applying a JAX-WS handler list to a WS-Notification service

To handle the messages that flow to and from an existing Version 7.0 WS-Notification service, you create JAX-WS handlers, chain them together in the form of a handler list, then apply the handler list to a NotificationBroker, PublisherRegistrationManager or SubscriptionManager endpoint at a Version 7.0 WS-Notification service point (for inbound invocation handling), or apply the handler list to a WS-Notification service (for outbound invocation handling).

Before you begin

This task assumes that you have already created a Version 7.0 WS-Notification service.

About this task

To create a JAX-WS handler, you can use a tool such as IBM Rational Application Developer. To enable handlers to perform more complex operations, you chain them together into handler lists. You associate each handler list with one or more Version 7.0 WS-Notification services or service points, so that the handler list can monitor WS-Notification activity and take appropriate action depending upon the sender and content of each inbound or outbound message. For example:

- You can use a handler list on the NotificationBroker Web service to log all notification messages received by this service point.
- You can use a handler list on a SubscriptionManager Web service to log all unsubscribe requests received by this service point.
- You can use a handler list on a PublisherRegistrationManager Web service to log all publisher deregistration requests received by this service point.

1. Create one or more JAX-WS handlers. You can do this using IBM Rational Application Developer or a similar tool.
2. Load JAX-WS handler classes. A JAX-WS handler interacts with messages through a Version 7.0 WS-Notification service point (for inbound invocation handling) or WS-Notification service (for outbound invocation handling), therefore you need to make the handler class available to the server or cluster that hosts the WS-Notification service point or service that you want to monitor.
3. Create a new JAX-WS handler configuration by using the administrative console or by using the createJAXWSHandler command. By creating a new handler configuration, you make WebSphere Application Server aware of your handler, and you make the handler available for inclusion in one or more handler lists.
4. Create a new JAX-WS handler list. The approach taken in WebSphere Application Server is to assign handler lists (rather than individual handlers) to WS-Notification service points (for inbound invocation handling) or WS-Notification services (for outbound invocation handling).
5. Optional: To apply a JAX-WS handler list to a service provider endpoint (NotificationBroker, PublisherRegistrationManager or SubscriptionManager) associated with a service point, use the administrative console to complete the following substeps:
   a. Navigate to **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the service points configured for this Version 7.0 WS-Notification service.
   b. In the content pane, click the name of a Version 7.0 WS-Notification service point in the list. The current settings for this Version 7.0 WS-Notification service point are displayed in the WS-Notification service points [Settings] form.
   c. Apply the JAX-WS handler list by selecting it from the list box for one or more of the following general properties:

      NotificationBroker JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the NotificationBroker endpoint of the WS-Notification service point.

      SubscriptionManager JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the SubscriptionManager endpoint of the WS-Notification service point.

      PublisherRegistrationManager JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the PublisherRegistrationManager endpoint of the WS-Notification service point.
6. Optional: To apply a JAX-WS handler list to a WS-Notification service, use the administrative console to complete the following substeps:
   a. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
   b. In the content pane, click the name of a Version 7.0 WS-Notification service in the list. The current settings for this Version 7.0 WS-Notification service are displayed in the WS-Notification services [Settings] panel.
   c. Apply the JAX-WS handler list by selecting it from the list box for the following general property:

      JAX-WS handler list

      The JAX-WS handler list that is applied to outbound requests from the WS-Notification service.

Configuring a Version 7.0 WS-Notification service with Web service QoS

You use the administrative console to configure Web service qualities of service (QoS) such as reliability or security by applying policy sets to a Version 7.0 WS-Notification service. The configuration of policy sets for WS-Notification can be split into two types: service provider (for Version 7.0 WS-Notification service points) and service client (for Version 7.0 WS-Notification service clients). Policy set configuration for these two elements of a WS-Notification service is handled differently.

Before you begin

This task assumes that you have already created a Version 7.0 WS-Notification service. You must also have configured policy sets that meet your quality of service requirements. You can reuse existing policy sets within your organization, use the default policy sets provided by WebSphere Application Server, or create new policy sets. For more information, see Managing policy sets using the administrative console.

About this task

Version 7.0 WS-Notification service points (NotificationBrokers, PublisherRegistrationManagers and SubscriptionManagers) are implemented as JAX-WS applications that are deployed to servers or clusters, and the management of the policy sets for these WS-Notification service points is configured using the policy set administrative infrastructure for service providers (both panels and wsadmin commands). In the Service provider [settings] panel for a WS-Notification service provider, there is a link to the associated WS-Notification service point application. You can use the service provider settings panel to attach policy sets to each NotificationBroker, PublisherRegistrationManager and SubscriptionManager.

Version 7.0 WS-Notification services are implemented as two configurable service clients (OutboundNotificationService and OutboundRemotePublisherService) for each WS-Notification service. Events that need outgoing Web service invocations occur at the service integration bus level rather than the bus member level, even though notifications of the events occur within a particular bus member. The two service clients that a WS-Notification service implements are therefore configured using the policy set administrative infrastructure for service clients. In the Service client policy sets and bindings [collection] panel, the two service clients for a given WS-Notification service are listed in a tree structure, along with their endpoints and operations. You can use this panel to attach policy sets to each service client, or to both service clients for the WS-Notification service.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
3. In the content pane, click the name of the Version 7.0 WS-Notification service to which to apply Web service qualities of service. The current settings for this Version 7.0 WS-Notification service are displayed in the WS-Notification services [Settings] panel.
4. Optional: To configure Web service QoS for inbound requests, apply policy sets to the service provider applications (NotificationBroker, PublisherRegistrationManager and SubscriptionManager) associated with a service point:

   Click **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the service points configured for this Version 7.0 WS-Notification service.

   Repeat the following steps, as required, to configure the policy set and binding information for one or more service provider applications for one or more service points:

   a. In the content pane, click the name of a Version 7.0 WS-Notification service point in the list. The current settings for this Version 7.0 WS-Notification service point are displayed in the WS-Notification service points [Settings] form.

   b. Click **[Additional Properties] Policy set configuration.** A link to each of the service provider applications (NotificationBroker, PublisherRegistrationManager and SubscriptionManager) for this service point is displayed in the Service provider [Collection] form.

c. Click the name of one of the applications. An indented list of endpoints and operations for this service is displayed in the Service provider [settings] form.
d. Select one or more items in the list (service, endpoint or operations), then perform one of the three available operations on the items you have selected:
   - Attach policy set
   - Detach policy set
   - Assign binding

   For more information, see Managing policy sets using the administrative console.
e. Restart the service provider application.

   Note: When you modify an existing policy set that has already been associated with a WS-Notification service point, there is no need to restart the service provider application; this is done automatically by the policy set infrastructure.

5. Optional: To configure Web service QoS for outbound requests, apply policy sets to one or both of the Web service clients associated with the WS-Notification service:
   a. If necessary, navigate back to the WS-Notification services [Settings] panel for this Version 7.0 WS-Notification service.
   b. Click **[Additional Properties] Outbound request policy sets and bindings**. The two service clients for this WS-Notification service are listed in a tree structure, along with their endpoints and operations, in the Service client policy set and bindings [collection] form.
   c. Select one or both service clients, then perform one of the three available operations on the items you have selected:
      - Attach client policy set
      - Detach client policy set
      - Assign binding

      For WS-Notification service clients, policy set attachments are not supported at the endpoint (port) or operation level. Therefore endpoints or operations are not selectable, and are shown as inheriting any policy set or binding that is attached to the service client. For more information, see Managing policy sets using the administrative console.

      Note: Using the Service client policy set and bindings [collection] form, you can configure the policy set and binding information for both service clients for the selected service. Alternatively, you can configure the policy set and binding information for a single Version 7.0 WS-Notification service client by clicking **Services** → **Service clients** → ***ws-notification_service_client_name*** and using the WS-Notification service client [settings] panel. This panel also provides links to the associated service integration bus and WS-Notification service.
   d. Restart the application server.

Results

The WS-Notification service has been successfully enabled with the required Web service QoS.

Configuring WS-Notification for reliable notification

To ensure that WS-Notification Web service interactions are performed in a reliable way, you configure a Version 7.0 WS-Notification service, JAX-WS client, and JAX-WS based WS-Notification consumer Web service (through policy set functionality) to use WS-ReliableMessaging.

Before you begin

This task assumes that you have created a Version 7.0 WS-Notification service and service point, and that you have created two JAX-WS applications:

- A JAX-WS client application, created from the WSDL of the new service point.
- A JAX-WS based WS-Notification consumer Web service.

For more information about how to create these applications, see “Using JAX-WS clients and Web services with new Version 7.0 WS-Notification service points” on page 1076.

About this task

Reliable notification refers to the reliable transmission of messages to and from the IBM WS-Notification implementation. You enable this reliability to mitigate the problems inherent in network transmission protocols such as HTTP.

Note: This reliability does not include the behavior of the application server itself. For more information, see WS-Notification and end-to-end reliability.

To enable reliable notification, you apply policy sets that include the WS-ReliableMessaging policy to the service point, service client, and service consumer applications.

You can configure policy sets for JAX-WS clients for both application server and client environments, including thin clients. For more information, see Managing policy sets using the administrative console.

1. Configure policy sets that meet your reliable messaging requirements.

   You can reuse existing policy sets within your organization, use the WS-ReliableMessaging default policy sets provided by WebSphere Application Server, or create new policy sets. For more information, see Configuring a WS-ReliableMessaging policy set using the administrative console.
2. Configure the Version 7.0 WS-Notification service and service points for reliable notification.

   Apply policy sets that include the WS-ReliableMessaging policy.
3. Configure the JAX-WS based WS-Notification client application so that it interacts reliably with its target Web service using WS-ReliableMessaging.

   Apply policy sets that include the WS-ReliableMessaging policy, as described in “Configuring a Version 7.0 WS-Notification service with Web service QoS” on page 1070 and Attaching and binding a WS-ReliableMessaging policy set to a Web service application.
4. Configure the JAX-WS based WS-Notification consumer Web service application so that it interacts reliably with clients that attempt to communicate with it.

   Apply policy sets that include the WS-ReliableMessaging policy, as described in Attaching and binding a WS-ReliableMessaging policy set to a Web service application.
5. Configure the WS-Notification client application (which has been configured to interact reliably) to communicate with the WS-Notification service point (which has similarly been configured to receive messages reliably).
6. Prompt the WS-Notification client application to subscribe on behalf on the (reliably configured) WS-Notification consumer Web service.

   Note: The WS-Notification consumer Web service might also act as a client to perform its own subscription. This client would require additional policy set configuration if reliable interactions were required.
7. Initiate notification message publications from the WS-Notification client application.

Results

The Version 7.0 WS-Notification service point receives the notification messages in a reliable way from the WS-Notification client, and publishes the notification messages in a reliable way to the WS-Notification consumer Web service.

Migrating a Version 6.1 WS-Notification configuration from WebSphere Application Server Version 6.1 to Version 7.0

You can migrate an existing Version 6.1 WS-Notification configuration to run in a WebSphere Application Server Version 7.0 environment.

Before you begin

This topic assumes that you have the following configuration:

- An existing server or cluster installation of WebSphere Application Server Version 6.1 that is ready for migration, including at least one WS-Notification service, service point, and underlying service integration bus.
- Existing subscriptions with associated messages pending notification.
- Existing operational WS-Notification client and Web service applications that are known to transmit and receive notifications through the existing Version 6.1 WS-Notification service.

About this task

When you migrate your system from WebSphere Application Server Version 6.1 to Version 7.0, any existing WS-Notification configuration (for example WS-Notification services, service points, administered subscribers), and the associated service integration bus resources (for example inbound services and endpoint listeners) are automatically migrated to valid configuration elements within the new installation. The services and service points are migrated to Version 6.1 WS-Notification services and service points, the endpoint URLs for the service points are unchanged, and the functionality supported is exactly the same as in WebSphere Application Server Version 6.1. For example, reliable notification is not possible with Version 6.1 WS-Notification services and service points.

Any messages pending notification are delivered following the update and server startup sequence.

1. Migrate your product configuration from WebSphere Application Server Version 6.1 to Version 7.0.
2. Check that notifications are being published and consumed correctly in your migrated system.

   Initiate the program, process, or application that includes the WS-Notification client (or clients), and that causes notifications to be requested and transmitted.

Results

The WS-Notification services and resources are migrated to Version 6.1 WS-Notification services and resources that run under WebSphere Application Server Version 7.0. Any messages pending notification are delivered.

What to do next

You are now ready (optionally) to prepare your migrated Version 6.1 WS-Notification configuration for reliable notification.

Preparing a migrated Version 6.1 WS-Notification configuration for reliable notification

You can gradually introduce JAX-WS based client and provider entities such that a migrated Version 6.1 WS-Notification configuration is ready to be configured for reliable notification.

Before you begin

This topic assumes that you have an existing server or cluster installation of WebSphere Application Server Version 7.0, including at least one WS-Notification service, service point, and underlying service integration bus that has been migrated to this version of the product as described in “Migrating a Version 6.1 WS-Notification configuration from WebSphere Application Server Version 6.1 to Version 7.0.”

About this task

For reliable notification, you apply policy sets that include WS-ReliableMessaging to your WS-Notification configuration. You can only use policy sets with Java API for XML-based Web services (JAX-WS) applications, and with Version 7.0 WS-Notification services and service points.

The WS-Notification implementation in WebSphere Application Server Version 6.1 uses service integration bus-enabled Web services to expose the WS-Notification service endpoint, so that it can be invoked by applications and configured with specific attributes such as WS-Security or JAX-RPC handlers. However, the Version 6.1 implementation is not compatible with JAX-WS handlers or applications, and it cannot compose with WS-ReliableMessaging.

To prepare a migrated Version 6.1 WS-Notification configuration for reliable notification, you need to recreate your Version 6.1 WS-Notification services and service points as Version 7.0 WS-Notification services and service points, and recreate each JAX-RPC client application to which you want to apply a policy set as a JAX-WS application. Note that you can continue to use JAX-RPC applications with Version 7.0 WS-Notification services and service points, and that you only need to recreate those applications that need to work with policy sets.

For information about coding JAX-RPC and JAX-WS client applications to perform specific WS-Notification tasks, see Developing applications that use WS-Notification. You might also find it useful to learn about JAX-WS and the JAX-WS client programming model. This should help you to determine the effort involved in porting client code from JAX-RPC to JAX-WS, or to validate JAX-WS client to JAX-RPC Web service interoperability.

To support a phased approach to preparing for reliable notification, and to describe the four main configurations that you might want to achieve, this task is divided into four subtasks:

- "Using JAX-WS clients and Web services with migrated service points."
- "Using JAX-RPC clients and Web services with new Version 7.0 WS-Notification service points" on page 1075.
- "Using JAX-WS clients and Web services with new Version 7.0 WS-Notification service points" on page 1076.
- "Sharing notifications between Version 6.1 and Version 7.0 WS-Notification service points" on page 1077.

What to do next

When you have completed these subtasks, you have a collection of WS-Notification client and server entities that are prepared for reliable notification, and you are ready to configure WS-Notification for reliable notification.

Using JAX-WS clients and Web services with migrated service points:

1. Publish notification messages through a migrated Version 6.1 WS-Notification service point, from a JAX-WS client application.
   a. Create a JAX-WS WS-Notification client application using the WSDL of the migrated service point.

      For more information, see Example: Publishing a WS-Notification message, Developing a JAX-WS client from a WSDL file and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.
   b. Run the application.
   c. Initiate one or more notification messages.

   The system accepts and publishes the notification messages from the JAX-WS client.
2. Receive notification messages in a new JAX-WS based WS-Notification consumer application, from a migrated Version 6.1 WS-Notification service point.

This validates that your Version 6.1 WS-Notification service point can deliver notifications to a JAX-WS consumer Web service.

a. Create a new JAX-WS based WS-Notification consumer Web service from the standard WS-Notification WSDL.

   For more information, see Example: Subscribing a WS-Notification consumer, Developing Web services applications from existing WSDL files with JAX-WS and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.

b. Create a subscription for the new consumer service through the Version 6.1 WS-Notification service point.

c. Prompt the WS-Notification service point to generate notifications (for example using a WS-Notification client application).

The system transmits the notifications to the new JAX-WS consumer application correctly.

Using JAX-RPC clients and Web services with new Version 7.0 WS-Notification service points:

1. Create a new Version 7.0 WS-Notification service.

   You can configure a Version 7.0 WS-Notification service and service points with policy sets to compose with WS-ReliableMessaging for reliable notification. The system creates and configures a new Version 7.0 WS-Notification service. This includes the creation of a Version 7.0 WS-Notification service point that exposes the service from a particular service integration bus member. Version 6.1 and Version 7.0 WS-Notification service points can coexist in WebSphere Application Server Version 7.0.

2. Publish notification messages through the new Version 7.0 WS-Notification service point, from a JAX-RPC client application.

   This validates the behavior of the Version 7.0 WS-Notification service point.

   a. Create the application using the WSDL of the new Version 7.0 WS-Notification service point.

      For more information, see Example: Publishing a WS-Notification message, Developing client bindings from a WSDL file for a JAX-RPC client and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.

      Note: Instead of creating a new JAX-RPC client application, you might choose to update an existing JAX-RPC client application from the WSDL of the new service point. The WSDL for a Version 7.0 WS-Notification service point contains a number of minor changes compared to a Version 6.1 service point, so you need to modify your existing JAX-WS client application to take account of these changes. Specifically, you need to regenerate the java proxy classes from the WSDL, and update any use of class names and methods that have changed. For example, there might be changes in the generated classes that include a port type or service from the WSDL.

   b. Run the application.

   c. Initiate one or more notification messages.

   The system accepts and publishes the notification messages from the JAX-RPC client.

3. Receive notification messages in a JAX-RPC based WS-Notification consumer application, from the new Version 7.0 WS-Notification service point.

   This validates that your Version 7.0 WS-Notification service point can deliver notifications to a JAX-RPC consumer Web service.

   a. Create a new JAX-RPC based WS-Notification consumer Web service from the standard WS-Notification WSDL.

      For more information, see Example: Subscribing a WS-Notification consumer, Developing Web services applications from existing WSDL files with JAX-WS and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.

Note: Instead of creating a new JAX-RPC consumer application, you can use an existing JAX-RPC consumer application from (for example) a Version 6.1 WS-Notification configuration.

b. Create a subscription for the new consumer service through the new Version 7.0 WS-Notification service point.

c. Prompt the WS-Notification service point to generate notifications (for example using a WS-Notification client application).

The system transmits the notifications to the new JAX-RPC consumer application correctly.

Using JAX-WS clients and Web services with new Version 7.0 WS-Notification service points:
Before you begin

Note that with this configuration you can compose with policy sets for reliable notification.

1. Publish notification messages through the new Version 7.0 WS-Notification service point, from a JAX-WS client application.

   a. Create a JAX-WS WS-Notification client application using the WSDL of the new Version 7.0 WS-Notification service point.

      For more information, see Example: Publishing a WS-Notification message, Developing a JAX-WS client from a WSDL file and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.

      Note: Instead of creating a new JAX-WS client application, you might choose to update the JAX-WS client application that you created in the subtask "Using JAX-WS clients and Web services with migrated service points" on page 1074. The WSDL for a Version 7.0 WS-Notification service point contains a number of minor changes compared to a Version 6.1 service point, so you need to modify your existing JAX-WS client application to take account of these changes. Specifically, you need to regenerate the java proxy classes from the WSDL, and update any use of class names and methods that have changed. For example, there might be changes in the generated classes that include a port type or service from the WSDL.

   b. Run the application.

   c. Initiate one or more notification messages.

   The system accepts and publishes the notification messages from the JAX-WS client.

2. Receive notification messages in a new JAX-WS based WS-Notification consumer application, from a new Version 7.0 WS-Notification service point.

   This validates that your Version 7.0 WS-Notification service point can deliver notifications to a JAX-WS consumer Web service.

   a. Create a new JAX-WS based WS-Notification consumer Web service from the standard WS-Notification WSDL.

      For more information, see Example: Subscribing a WS-Notification consumer, Developing Web services applications from existing WSDL files with JAX-WS and "Publishing the WSDL files for a WS-Notification application to a ZIP file" on page 1079.

      Note: Instead of creating a new JAX-WS consumer application, you might choose to update the JAX-WS consumer application that you created in the subtask "Using JAX-WS clients and Web services with migrated service points" on page 1074. The WSDL for a Version 7.0 WS-Notification service point contains a number of minor changes compared to a Version 6.1 service point, so you need to modify your existing JAX-WS client application to take account of these changes. Specifically, you need to regenerate the java proxy classes from the WSDL, and update any use of class names and methods that have changed. For example, there might be changes in the generated classes that include a port type or service from the WSDL.

b. Create a subscription for the new consumer service through the new Version 7.0 WS-Notification service point.
c. Prompt the WS-Notification service point to generate notifications (for example using a WS-Notification client application).

The system transmits the notifications to the new JAX-WS consumer application correctly.

Sharing notifications between Version 6.1 and Version 7.0 WS-Notification service points:
About this task

You can configure WS-Notification so that notifications received through migrated Version 6.1 WS-Notification service points are published through the new Version 7.0 service. You might want to do this so that (for example) you can receive notifications through existing, unreliable connections then publish them through new connections made reliable through WS-ReliableMessaging. To enable this configuration, the new Version 7.0 WS-Notification service needs to use the same service integration bus topic space as the migrated Version 6.1 WS-Notification service. You use a permanent topic namespace to statically define the association between a WS-Notification topic namespace URI and a service integration bus topic space destination. You configure a permanent topic namespace as a property of a WS-Notification service.

1. Discover which bus topic spaces the migrated Version 6.1 WS-Notification service is using. If none, create a new permanent topic namespace to connect to a bus topic space. For more information, see “Modifying a Version 6.1 WS-Notification service” on page 1092.
2. Create a new permanent topic namespace for the new Version 7.0 WS-Notification service, that connects to the same bus topic space. For more information, see “Modifying a Version 7.0 WS-Notification service” on page 1085.

Results

Notifications received by either the new or migrated service point are now published to subscriptions made on either WS-Notification service.

Interacting at run time with WS-Notification

View (and in some cases delete) at run time the active items associated with WS-Notification service points.

Before you begin

This task assumes that you have a fully configured and operational WS-Notification service point.

About this task

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

You can use the administrative console to interact at run time with the following active items associated with WS-Notification service points:

Publisher registrations
A runtime list of existing publisher registrations is provided. This includes, for each publisher registration, the information that was used to create it and when it will terminate.

Pull points
A runtime list is provided of the pull points that have been created. This includes, for each pull point, its current termination time and a link to the associated subscription.

Subscriptions
A runtime view is provided for subscriptions that have been created by applications. This includes, for each subscription, the information that was used to create it and an indication of its current state.

Administered subscribers
A runtime list is provided of the administered subscribers for a given WS-Notification service point. This includes, for each administered subscriber, an indication of its current state; for example whether the subscription was successfully initialized at start time.

For each WS-Notification service point, runtime information is available for subscriptions, registrations, pull points and administered subscribers. **For each WS-Notification service**, runtime information is available - aggregated for all service points for the service - for subscriptions, registrations and pull points. There is no aggregated view for administered subscribers at the WS-Notification service level.

To access and use the runtime information for active items associated with WS-Notification service points, use the administrative console to complete the following steps:

1. Choose between information for a particular service point, or information aggregated for all service points for a particular service, by completing one of the following substeps:
   a. Optional: For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.
   b. Optional: For runtime information aggregated for all service points for a particular service, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations and pull points for this WS-Notification service.
2. Click **Subscriptions** , **Publisher registrations**, **Pull points** or (for a particular service point) **Administered subscribers**.
   - If you click **Subscriptions**, a panel is displayed that lists the durable subscriptions that have been created by a WS-Notification service point in response to “Subscribe” requests from WS-Notification applications. You can view the subscription name (ID), topic and other information associated with the subscription, and you can view messages held on the durable subscription pending delivery. You can also delete subscriptions.

     Note: Both the WS-Notification subscription and publisher registration resources include the concept of scheduled termination, in which the application indicates a period of time after which the resource is destroyed. “Badly-behaved” in this case describes an application that requests an infinite lifetime for a resource and then does not explicitly delete the resource before it goes away (never to return).
   - If you click **Publisher registrations**, a panel is displayed that lists the publisher registrations that are currently in effect on this WS-Notification service or service point (that is, applications that have registered as publishers). You can view the basic properties of the registration record. You can also delete a publisher registration record.
   - If you click **Pull points**, a panel is displayed that lists the pull points that are currently active on this WS-Notification service or service point. You can view basic properties of the pull point such as the

subscription with which it is associated and the time at which it is currently set to expire, and you can navigate to the associated subscriptions where appropriate. You can also delete pull points.

- For a particular service point, if you click **Administered subscribers**, a panel is displayed that lists the administered subscribers that are currently in effect on this WS-Notification service point. You can use this information to see whether a given subscriber has been successfully initialized.

What to do next

For more detailed information about working with individual runtime panels, see the following topics:

- "Listing or deleting active WS-Notification subscriptions" on page 1119.
- "Listing or deleting active WS-Notification publisher registrations" on page 1121.
- "Listing or deleting active WS-Notification pull points" on page 1122.
- "Listing active WS-Notification administered subscribers" on page 1123.

Publishing the WSDL files for a WS-Notification application to a ZIP file

Use the administrative console to download a zip file that contains a WS-Notification application's published WSDL files.

About this task

The ability to publish these WSDL files to a ZIP file is particularly useful in the following circumstances:

- Writing a WS-Notification application that invokes Web service operations against the NotificationBroker application, as described in Writing a WS-Notification application that does not expose a Web service endpoint.
- Viewing the endpoint URLs to which WS-Notification applications connect, by looking in the WSDL file for the NotificationBroker application for Version 7.0 services, or the inbound service for Version 6.1 services.

1. Start the administrative console.
2. Navigate to the "Publish WSDL files to ZIP file [Settings]" form for the WS-Notification application.

   For Version 7.0 WS-Notification services, click one of the following paths:

   - **Service integration → WS-Notification → Services → *service_name* → [Additional Properties] WS-Notification service points → *point_name* → [Additional Properties] Publish WSDL files to zip**
   - **Service integration → Buses → *bus_name* → [Services] WS-Notification services → *service_name* → [Additional Properties] WS-Notification service points → *point_name* → [Additional Properties] Publish WSDL files to zip**

   For Version 6.1 WS-Notification services, click one of the following paths:

   - **Service integration → WS-Notification → Services → *service_name* → [Related Items] Notification broker inbound service settings → [Additional Properties] Publish WSDL files to ZIP file**
   - **Service integration → Buses → *bus_name* → [Services] WS-Notification services → *service_name* → [Related Items] Notification broker inbound service settings → [Additional Properties] Publish WSDL files to ZIP file**
3. Click on the file name to download a zip file that contains the application's published WSDL files.

Configuring WS-Notification resources

You can configure individual WS-Notification resources by using the administrative console or by using the wsadmin tool.

About this task

To complete the tasks described in "Accomplishing common WS-Notification tasks" on page 1057, you configure the following types of WS-Notification resource:

- WS-Notification service
- WS-Notification service point
- WS-Notification administered subscriber
- Permanent WS-Notification topic namespace
- WS-Notification topic namespace document

You can configure individual WS-Notification resources by using the administrative console or by using the wsadmin tool, as described in the following tasks:

- "Configuring WS-Notification resources by using the administrative console."
- WSNotificationCommands command group for the AdminTask object.
- "Interacting at run time with WS-Notification" on page 1077.

Configuring WS-Notification resources by using the administrative console

Use the administrative console to configure individual WS-Notification services, service points, administered subscribers, permanent topic namespaces, topic namespace documents and JAX-WS handlers.

Before you begin

Decide on which method you want to use to configure these resources. You can configure WS-Notification resources by using the administrative console as described in this task, or by using the commands in the WSNotificationCommands command group for the AdminTask object.

About this task

WS-Notification enables Web services to use the "publish and subscribe" messaging pattern. For more information, see WS-Notification: Overview and "Learning about WS-Notification" on page 1059.

For high-level configuration of WS-Notification, see "Accomplishing common WS-Notification tasks" on page 1057. Use the steps for this task to configure individual instances of each type of WS-Notification resource.

You access the WS-Notification panels through the **Service integration** section of the administrative console navigation pane:

- To see a list of all the WS-Notification services, click **Service integration** → **WS-Notification** → **Services**.
- To see a list of all the WS-Notification services for a particular service integration bus, click **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**.

To support the Java API for XML-based Web services (JAX-WS) and composition with WS-ReliableMessaging, you create your WS-Notification services as JAX-WS applications. The implementation of WS-Notification in WebSphere Application Server Version 6.1 uses JAX-RPC applications, so there are now two different implementations of the WS-Notification service and service point:

- **Version 7.0**: You configure a Version 7.0 WS-Notification service and service points if you want to compose a JAX-WS WS-Notification service with WS-ReliableMessaging, or if you want to apply JAX-WS handlers to your WS-Notification service. This is the recommended type of service for new deployments.
- **Version 6.1**: You configure a Version 6.1 WS-Notification service and service points if you want to expose a JAX-RPC WS-Notification service using the same technology provided in WebSphere Application Server Version 6.1, including the ability to apply JAX-RPC handlers to the service.

To configure specific WS-Notification resources, use the administrative console to complete any of the following tasks:

- Configure a WS-Notification service.

  Complete any of the following tasks:
  - Create a new Version 7.0 WS-Notification service.
  - Modify a Version 7.0 WS-Notification service.
  - Create a new Version 6.1 WS-Notification service.
  - Modify a Version 6.1 WS-Notification service.
  - Delete WS-Notification services.
- Configure a WS-Notification service point.

  Complete any of the following tasks:
  - Create a new Version 7.0 WS-Notification service point.
  - Modify a Version 7.0 WS-Notification service point.
  - Create a new Version 6.1 WS-Notification service point.
  - Modify a Version 6.1 WS-Notification service point.
  - Delete WS-Notification service points.
- Configure a WS-Notification administered subscriber.

  Complete any of the following tasks:
  - Create a new WS-Notification administered subscriber.
  - Modify a WS-Notification administered subscriber.
  - Delete WS-Notification administered subscribers.
- Configure a permanent WS-Notification topic namespace.

  Complete any of the following tasks:
  - Create a new permanent WS-Notification topic namespace.
  - Show the properties of a permanent WS-Notification topic namespace.
  - Delete WS-Notification permanent topic namespaces.
- Configure a WS-Notification topic namespace document.

  Complete any of the following tasks:
  - Apply a WS-Notification topic namespace document.
  - Show the contents of a WS-Notification topic namespace document.
  - Delete WS-Notification topic namespace documents.
- Configure JAX-WS handlers and handler lists.

  Complete any of the following tasks:
  - Load JAX-WS handler classes.
  - Create a new JAX-WS handler configuration.
  - Modify an existing JAX-WS handler configuration.
  - Delete JAX-WS handler configurations.
  - Create a new JAX-WS handler list.
  - Modify an existing JAX-WS handler list.
  - Delete JAX-WS handler lists.

Creating a new Version 7.0 WS-Notification service:

Create a new WS-Notification service and the associated objects that form the infrastructure of the WS-Notification configuration. You configure a Version 7.0 WS-Notification service and service points if you

want to compose a JAX-WS WS-Notification service with WS-ReliableMessaging, or if you want to apply JAX-WS handlers to your WS-Notification service. This is the recommended type of service for new deployments.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new Version 7.0 WS-Notification service by using the administrative console as described in this task, or by using the createWSNService command.

This task assumes that you have an existing service integration bus, configured with at least one bus member.

You usually configure one WS-Notification service for a service integration bus, but you can configure more than one. For more information, see Reasons to create multiple WS-Notification services in a bus.

Defining a Version 7.0 WS-Notification service is not the same as exposing a NotificationBroker (WSDL) port to which Web services applications can connect. To do this, create one or more Version 7.0 WS-Notification service points as described in this task.

About this task

A WS-Notification service provides the ability to expose some or all of the messaging resources defined on a service integration bus for use by WS-Notification applications.

To support the Java API for XML-based Web services (JAX-WS) and composition with WS-ReliableMessaging, you create your WS-Notification services as JAX-WS applications, then use this task to create a Version 7.0 WS-Notification service, one or more service points, and (optionally) a permanent topic namespace.

You can also apply JAX-WS handler lists to WS-Notification service points (for inbound invocation handling) and WS-Notification services (for outbound invocation handling).

When you create a Version 7.0 WS-Notification service, the wizard creates and deploys a JAX-WS based provider application. This application exposes the WS-Notification Web service interfaces for each of the three WS-Notification service roles:

- Notification broker
- Subscription manager
- Publisher registration manager

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
3. In the content pane, click **New**. The “New WS-Notification service” wizard is displayed. For more information about the properties that you set with the wizard, see WS-Notification services [Settings].
4. Step 1: Configure name, description, service integration bus and dynamic topic namespace settings.
   a. Enter your chosen name and an optional description.

      The name forms part of the endpoint on which the service is exposed (that is, the URL used to access the WS-Notification service points that are defined under the service). For Version 6.1 WS-Notification services, the service name is unique within a bus. For Version 7.0 WS-Notification services, the service name is unique within the cell - which matches the administration model used for policy sets, and therefore supports composition of Version 7.0 WS-Notification services with WS-ReliableMessaging.

b. Select or deselect the option **Enable dynamic topic namespaces?**.

Indicates whether dynamic topic namespaces can be used within the WS-Notification service. That is, whether this service allows dynamic topic namespaces to be created at run time. For more information, see Dynamic topic namespace.

Use this option to tightly control the topic namespaces that are used when connecting to a particular WS-Notification service (for example for security or auditing requirements). If you deselect this option, any applications that connect to the WS-Notification service and request topics from a dynamic topic namespace are stopped from publishing or receiving messages.

All messages published to a dynamic topic namespace are inserted with the default message reliability setting of `reliable persistent`. If this value is not acceptable, create a permanent topic namespace and manually configure the attribute to the appropriate value.

Note: The dynamic topic namespaces used on a particular WS-Notification service are backed by a service integration bus topic space that is created automatically when you create the topic namespace. The syntax of topics used within this topic space is internal to the WS-Notification service implementation.

c. Select or deselect the option **Requires registration**.

Indicates whether publisher applications are required to register with the broker before they can publish notifications.

d. Select a service integration bus from the selection list.

e. Click **Next**.

5. Step 2: Select WS-Notification service type.

Select **Version 7.0** as the type of service that you want to create.

6. Step 3: Configure handler and web service policy settings.

These settings are applied to the event notifications exchanged with WS-Notification client applications.

a. Optional: Choose a JAX-WS handler list.

The JAX-WS handler list that is applied to outbound requests from the WS-Notification service. A handler list defines the handlers that are applied when making outbound Web service invocations, for example monitoring outbound event notification (in response to a subscribe operation) and controlling demand-based publishers (subscribe, pause and resume). For more information about handler lists, see “Configuring JAX-WS handlers” on page 1068.

b. Enable or clear the **Query WSDL** option.

Indicates whether or not the Version 7.0 WS-Notification service queries the WSDL of other WS-Notification Web services when interacting with them. By default, this option is enabled. By clearing this option, you can improve performance by avoiding expensive WSDL queries. However, please note the following considerations when WSDL querying is not enabled:

- WS-Notification attempts to discover binding information (which is normally discovered through the WSDL) using other means. WS-Notification uses the SOAP version associated with the WS-Notification service point where subscriptions were made (by other Web services), or where administered subscriptions were created (by an administrator).
- There are some circumstances in which WS-Notification might be unable to determine binding information. This can happen when cleaning up subscriptions where the associated service point has been deleted and configuration information is no longer available. Under these circumstances WS-Notification makes a “best guess” at binding information to use to clean up the subscriptions.
- There is one scenario where incorrect binding information is used. That is, when a subscriber subscribes using a particular SOAP binding, on behalf of a NotificationConsumer that expects notifications using a different SOAP binding.

c. Enter a dynamic topic space name.

The name of the service integration bus topic space to be used as the dynamic topic space for this WS-Notification service. That is, the name of the bus topic space that is used to host the ad-hoc topic namespace, and to host dynamic topic namespaces if they are permitted. A default name of `WSN_dynamic_this_service_name` is offered.

d. Click **Next**.

7. Step 4: Create WS-Notification service points.

   A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service. For more information, see WS-Notification service point.

   a. Select **Yes** to create a new WS-Notification service point, then click **Next**.

      A WS-Notification service must have at least one service point.

   b. Supply a name and (optional) description for the WS-Notification service point, and from the selection list select the bus member on which the service point is to be configured, then click **Next**.

      The service point name forms part of the URL used to access the service point. On a single server system there is only one bus member in the list.

   c. Select the transport settings for the new service point.

      Service point accessed via HTTP proxy
      If the service point is accessed through a proxy, enable the check box, and type the root of the externally visible endpoint address URL for Web services accessed using this endpoint.

      The URL for the proxy is used to populate the WSDL endpoint address fields when publishing WSDL files to a zip file.

      SOAP Version
      Select the version of SOAP that is supported by the service point. This affects the WSDL definition that is exposed by the Web service.

   d. Optional: Select the JAX-WS handler list settings for the new service point.

      NotificationBroker JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the NotificationBroker endpoint of the WS-Notification service point.

      SubscriptionManager JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the SubscriptionManager endpoint of the WS-Notification service point.

      PublisherRegistrationManager JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the PublisherRegistrationManager endpoint of the WS-Notification service point.

   e. Click **Next**. The new service point is added to the list of service points for this WS-Notification service.

   f. Optional: To create another service point, repeat the previous substeps.

   g. When you have finished creating service points for this WS-Notification service, select **No** for the option to create another service point, then click **Next**.

8. Optional: Step 5: Create permanent topic namespaces.

For more information, see Permanent topic namespace. When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents.

a. Select **Yes** to create a new permanent topic namespace, then click **Next**.
b. Enter a name for the permanent topic namespace.

   This is the URI by which WS-Notification applications refer to topics hosted by this namespace.
c. Associate this new permanent topic namespace with the service integration bus topic space that you want to use to publish and receive messages.

   From the service integration bus topic space selection list, complete one of the following actions:
   - Choose the name of an existing bus topic space.
   - Choose the option to `Create a new topic space`, then enter a name for the new topic space.
d. Select from the selection list the service integration bus reliability (quality of service) that is assigned to messages published using this topic namespace.

   You can choose one of five values, each representing one of the service integration bus message reliability levels. The default value is `reliable persistent`, which is the value used by default for JMS Persistent messages.
e. Click **Next**.

   The new permanent topic namespace is added to a list of permanent topic namespaces for this Version 7.0 WS-Notification service, and you are asked whether you want to create another permanent topic namespace (default is **Yes**).
f. Optional: To create another permanent topic namespace, repeat the previous substeps.
g. When you have finished creating permanent topic namespaces for this Version 7.0 WS-Notification service, select **No** for the option to create another permanent topic namespace, then click **Next**.

9. Step 6: Summary.

   Check that the summary of the actions taken by the wizard is as you expected, then click **Finish**. If the processing completes successfully, the list of Version 7.0 WS-Notification services is updated to include the new Version 7.0 WS-Notification service. Otherwise, an error message is displayed.
10. Save your changes to the master configuration.
11. Optional: Restart the server if either of the following conditions apply:
    - A new bus or new bus member has been created as part of this task.
    - **Configuration reload** is not enabled for the bus.

What to do next

To perform advanced configuration tasks for this WS-Notification service (for example, adding additional service points and applying topic namespace documents to permanent topic namespaces), see “Modifying a Version 7.0 WS-Notification service.”

To perform advanced configuration tasks for the WS-Notification service point that you created as part of this task (for example, adding administered subscribers, publishing WSDL files to zip, and configuring the enterprise application associated with this service point), see “Modifying a Version 7.0 WS-Notification service point” on page 1097.

To configure this WS-Notification service or service point with Web service qualities of service (QoS) such as reliability or security, see “Configuring a Version 7.0 WS-Notification service with Web service QoS” on page 1070.

Modifying a Version 7.0 WS-Notification service:

Modify the **description**, **Enabled dynamic topic namespaces?**, **Requires registration**, **JAX-WS handler list** and **Query WSDL** properties of a Version 7.0 WS-Notification service, and follow links to complete advanced configuration such as adding additional WS-Notification service points, applying topic namespace documents to permanent topic namespaces, and applying policy sets to enable WS-ReliableMessaging.

About this task

A WS-Notification service provides the ability to expose some or all of the messaging resources defined on a service integration bus for use by WS-Notification applications.

A handler list defines the handlers that are applied when making outbound Web service invocations, for example monitoring outbound event notification (in response to a subscribe operation) and controlling demand-based publishers (subscribe, pause and resume).

When you create a Version 7.0 WS-Notification service, the wizard creates and deploys a JAX-WS based provider application. This application exposes the WS-Notification Web service interfaces for each of the three WS-Notification service roles:

- Notification broker
- Subscription manager
- Publisher registration manager

You can also configure custom properties to specify a timeout time for outbound requests, and to determine the strictness of the syntax checking of topics used under this Version 7.0 WS-Notification service.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
3. In the content pane, click the name of a Version 7.0 WS-Notification service in the list. The current settings for this Version 7.0 WS-Notification service are displayed in the WS-Notification services [Settings] panel.
4. Modify the following general properties:

   Description
   An optional description of the WS-Notification service.

   Enable dynamic topic namespaces?

   Indicates whether dynamic topic namespaces can be used within the WS-Notification service. That is, whether this service allows dynamic topic namespaces to be created at run time. For more information, see Dynamic topic namespace.

   Use this option to tightly control the topic namespaces that are used when connecting to a particular WS-Notification service (for example for security or auditing requirements). If you deselect this option, any applications that connect to the WS-Notification service and request topics from a dynamic topic namespace are stopped from publishing or receiving messages.

   All messages published to a dynamic topic namespace are inserted with the default message reliability setting of `reliable persistent`. If this value is not acceptable, create a permanent topic namespace and manually configure the attribute to the appropriate value.

   Note: The dynamic topic namespaces used on a particular WS-Notification service are backed by a service integration bus topic space that is created automatically when you create the topic namespace. The syntax of topics used within this topic space is internal to the WS-Notification service implementation.

Requires registration

Indicates whether publisher applications are required to register with the broker before they can publish notifications.

JAX-WS handler list

The JAX-WS handler list that is applied to outbound requests from the WS-Notification service.

A handler list defines the handlers that are applied when making outbound Web service invocations, for example monitoring outbound event notification (in response to a subscribe operation) and controlling demand-based publishers (subscribe, pause and resume). For more information about handler lists, see "Configuring JAX-WS handlers" on page 1068.

Query WSDL

Indicates whether or not the Version 7.0 WS-Notification service queries the WSDL of other WS-Notification Web services when interacting with them. By default, this option is enabled. By clearing this option, you can improve performance by avoiding expensive WSDL queries. However, please note the following considerations when WSDL querying is not enabled:

- WS-Notification attempts to discover binding information (which is normally discovered through the WSDL) using other means. WS-Notification uses the SOAP version associated with the WS-Notification service point where subscriptions were made (by other Web services), or where administered subscriptions were created (by an administrator).
- There are some circumstances in which WS-Notification might be unable to determine binding information. This can happen when cleaning up subscriptions where the associated service point has been deleted and configuration information is no longer available. Under these circumstances WS-Notification makes a "best guess" at binding information to use to clean up the subscriptions.
- There is one scenario where incorrect binding information is used. That is, when a subscriber subscribes using a particular SOAP binding, on behalf of a NotificationConsumer that expects notifications using a different SOAP binding.

5. Modify the additional properties:

WS-Notification service points

Select this link to configure the deployment of WS-Notification service points on one or more servers. For more information, see "Creating a new Version 7.0 WS-Notification service point" on page 1095 or "Modifying a Version 7.0 WS-Notification service point" on page 1097.

Permanent topic namespaces

Select this link to configure permanent topic namespaces for the WS-Notification service. For more information, see Permanent topic namespace. When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents.

Custom properties

Select this link to configure additional custom properties for this WS-Notification service. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value.

To specify a timeout time for outbound requests sent from this WS-Notification service, set the following custom property:

```
outbound.timeout
```

The value of this property is the timeout time in milliseconds. If the property is not set, a default timeout of 2 minutes is used.

To determine the strictness of the syntax checking of topics used under this WS-Notification service, set the following custom property:

`com.ibm.ws.sib.wsn.strictTopicChecking`

Valid values for this property are TRUE and FALSE:

- If the property value is set to TRUE, the topic syntax rules defined in the WS-Topics standard are strictly enforced. Note that there is a performance overhead compared to the default setting, because each character of a topic is validated against a large list of permitted Unicode characters.
- If the property is omitted or set to FALSE, syntax checking only ensures that the basic topic structure is valid, and character checking is relaxed to allow any character exception "*" and "." as a topic name part.

Outbound request policy sets and bindings

The outbound request policy sets and bindings for the two WS-Notification service clients associated with this WS-Notification service. For reliable Web service transmission of notification messages, use this option to associate the WS-Notification service client with a policy set that enables WS-ReliableMessaging.

For more information, see "Configuring a Version 7.0 WS-Notification service with Web service QoS" on page 1070.

6. Apply any changes, then click **OK**. If the processing completes successfully, the list of WS-Notification services is redisplayed. Otherwise, an error message is displayed.
7. Save your changes to the master configuration. You need not restart the server for the changes to fully take effect if **configuration reload** is enabled for the service integration bus.

Deleting WS-Notification services:

Remove all configuration associated with one or more WS-Notification services. A WS-Notification service provides access to service integration bus resources for Web services publish and subscribe clients.

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification service by using the administrative console as described in this task, or by using the deleteWSNService command.

About this task

When you delete a Version 6.1 WS-Notification service, the associated inbound services and inbound ports are also deleted. The associated endpoint listeners are deleted if they were created in the process of creating a WS-Notification service point and are not used by any other configuration object. When you delete a Version 7.0 WS-Notification service, the associated service point provider applications and service points are also deleted. Do not delete these resources manually, because this might leave the associated configuration information in an inconsistent state.

For both types of WS-Notification service, you choose whether or not the associated service integration bus topic spaces are deleted.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. A list of all the WS-Notification services is displayed in a WS-Notification services [Collection] form.
3. Select the check box for every WS-Notification service that you want to remove.

4. Click **Delete**. A panel is displayed asking if the service integration bus topic spaces associated with the WS-Notification service (including those associated with all the WS-Notification topic namespaces) should also be deleted.
5. Optional: Select the check box if you want to delete the associated service integration bus topic spaces.

   Note: Deleting a service integration bus topic space causes exception messages to be generated for WS-Notification applications that reference the topic space, as described in Failures as a result of changes in topic space and topic namespace configurations.
6. Click **Delete**. If the processing completes successfully, the list of WS-Notification services is updated. Otherwise, an error message is displayed.
7. Save your changes to the master configuration.

Creating a new Version 6.1 WS-Notification service:

Create a new WS-Notification service and the associated objects that form the infrastructure of the WS-Notification configuration. You configure a Version 6.1 WS-Notification service and service points if you want to expose a JAX-RPC WS-Notification service using the same technology provided in WebSphere Application Server Version 6.1, including the ability to apply JAX-RPC handlers to the service.

Before you begin

Ensure that you have successfully configured an SDO repository, as described in “Installing and configuring the SDO repository” on page 1003. The SDO repository is used to store WSDL documents during the creation of the WS-Notification service. If you do not configure the repository, an error message appears when you create the service.

Decide on which method you want to use to configure these resources. You can create a new Version 6.1 WS-Notification service by using the administrative console as described in this task, or by using the createWSNService command.

This task assumes that you have an existing service integration bus, configured with at least one bus member.

You usually configure one WS-Notification service for a service integration bus, but you can configure more than one. For more information, see Reasons to create multiple WS-Notification services in a bus.

Defining a WS-Notification service on a bus is not the same as exposing a NotificationBroker (WSDL) port to which Web services applications can connect. To do this, create one or more WS-Notification service points as described in this task.

About this task

A WS-Notification service provides the ability to expose some or all of the messaging resources defined on a service integration bus for use by WS-Notification applications.

A JAX-RPC handler list and WS-Security bindings define the parameters and security policy that are used when making outbound Web service invocations, for example monitoring outbound event notification (in response to a subscribe operation) and controlling demand-based publishers (subscribe, pause and resume).

When you create a Version 6.1 WS-Notification service, the wizard configures three service integration bus inbound services for the WS-Notification service, one for each of the three WS-Notification service roles:

- Notification broker
- Subscription manager

- Publisher registration manager

These inbound services are defined on the same service integration bus as the Version 6.1 WS-Notification service, and each of these inbound services refers to the same bus destination.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
3. In the content pane, click **New**. The "New WS-Notification service" wizard is displayed. For more information about the properties that you set with the wizard, see WS-Notification services [Settings].
4. Step 1: Configure name, description, service integration bus and dynamic topic namespace settings.
   a. Enter your chosen name and an optional description.

      The name forms part of the endpoint on which the service is exposed (that is, the URL used to access the WS-Notification service points that are defined under the service). For Version 6.1 WS-Notification services, the service name is unique within a bus. For Version 7.0 WS-Notification services, the service name is unique within the cell - which matches the administration model used for policy sets, and therefore supports composition of Version 7.0 WS-Notification services with WS-ReliableMessaging.
   b. Select or deselect the option **Enable dynamic topic namespaces?**.

      Indicates whether dynamic topic namespaces can be used within the WS-Notification service. That is, whether this service allows dynamic topic namespaces to be created at run time. For more information, see Dynamic topic namespace.

      Use this option to tightly control the topic namespaces that are used when connecting to a particular WS-Notification service (for example for security or auditing requirements). If you deselect this option, any applications that connect to the WS-Notification service and request topics from a dynamic topic namespace are stopped from publishing or receiving messages.

      All messages published to a dynamic topic namespace are inserted with the default message reliability setting of `reliable persistent`. If this value is not acceptable, create a permanent topic namespace and manually configure the attribute to the appropriate value.

      Note: The dynamic topic namespaces used on a particular WS-Notification service are backed by a service integration bus topic space that is created automatically when you create the topic namespace. The syntax of topics used within this topic space is internal to the WS-Notification service implementation.
   c. Select or deselect the option **Requires registration**.

      Indicates whether publisher applications are required to register with the broker before they can publish notifications.
   d. Select a service integration bus from the selection list.
   e. Click **Next**.
5. Step 2: Select WS-Notification service type.

   Select **Version 6.1** as the type of service that you want to create.
6. Step 3: Configure handler and web service policy settings.

   These settings are applied to the event notifications exchanged with WS-Notification client applications.
   a. Optional: Choose a JAX-RPC handler list.

      The JAX-RPC handler list that is applied to outbound requests from the WS-Notification service - for example the broker delivering notifications to a consumer. For more information about handler lists, see "Working with JAX-RPC handlers and clients" on page 1029.
   b. Optional: Choose a WS-Security configuration and bindings:

Outbound security request binding
The security binding to be used with consumer notifications and remote publisher requests sent by this WS-Notification service.

Outbound security response binding
The security binding to be used with remote publisher responses received by this WS-Notification service.

Outbound security configuration
Specifies the details of how security is applied to requests and responses.

For more information about Web services security resources, see Configuring secure transmission of SOAP messages using WS-Security.

c. Enter a dynamic topic space name.

The name of the service integration bus topic space to be used as the dynamic topic space for this WS-Notification service. That is, the name of the bus topic space that is used to host the ad-hoc topic namespace, and to host dynamic topic namespaces if they are permitted. A default name of `WSN_dynamic_this_service_name` is offered.

d. Click **Next**.

7. Step 4: Create WS-Notification service points.

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service. For more information, see WS-Notification service point.

a. Select **Yes** to create a new WS-Notification service point, then click **Next**.

A WS-Notification service must have at least one service point.

b. Supply a name and (optional) description for the WS-Notification service point, and from the selection list select the bus member on which the service point is to be configured, then click **Next**.

The service point name forms part of the URL used to access the service point (that is, the address of the Web service that is exposed on the chosen server). On a single server system there is only one bus member in the list.

c. Select a listener application to use to expose the service. Either select an existing endpoint listener for this bus member, or **Create a new endpoint listener**.

For more information, see “Creating a new endpoint listener configuration” on page 1019.

d. Click **Next**. The new service point is added to the list of service points for this WS-Notification service.

e. Optional: To create another service point, repeat the previous substeps.

f. When you have finished creating service points for this WS-Notification service, select **No** for the option to create another service point, then click **Next**.

8. Optional: Step 5: Create permanent topic namespaces.

When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents. For more information, see Permanent topic namespace.

a. Select **Yes** to create a new permanent topic namespace, then click **Next**.

b. Enter a name for the permanent topic namespace.

   This is the URI by which WS-Notification applications refer to topics hosted by this namespace.

c. Associate this new permanent topic namespace with the service integration bus topic space that you want to use to publish and receive messages.

   From the service integration bus topic space selection list, complete one of the following actions:

   - Choose the name of an existing bus topic space.
   - Choose the option to `Create a new topic space`, then enter a name for the new topic space.

d. Select from the selection list the service integration bus reliability (quality of service) that is assigned to messages published using this topic namespace.

   You can choose one of five values, each representing one of the service integration bus message reliability levels. The default value is `reliable persistent`, which is the value used by default for JMS Persistent messages.

e. Click **Next**.

   The new permanent topic namespace is added to a list of permanent topic namespaces for this WS-Notification service, and you are asked whether you want to create another permanent topic namespace (default is **Yes**).

f. Optional: To create another permanent topic namespace, repeat the previous substeps.

g. When you have finished creating permanent topic namespaces for this WS-Notification service, select **No** for the option to create another permanent topic namespace, then click **Next**.

9. Step 6: Summary.

   Check that the summary of the actions taken by the wizard is as you expected, then click **Finish**. If the processing completes successfully, the list of WS-Notification services is updated to include the new Version 6.1 WS-Notification service. Otherwise, an error message is displayed.

10. Save your changes to the master configuration.

11. Optional: Restart the server if either of the following conditions apply:

    - A new bus or new bus member has been created as part of this task.
    - **Configuration reload** is not enabled for the bus.

What to do next

To perform advanced configuration tasks for this WS-Notification service (for example adding additional WS-Notification service points, or applying topic namespace documents to permanent topic namespaces), see “Modifying a Version 6.1 WS-Notification service.”

Modifying a Version 6.1 WS-Notification service:

Modify the **description**, **Enabled dynamic topic namespaces?** and **Requires registration** properties of a WS-Notification service, and follow links to complete advanced configuration of the WS-Notification service such as adding additional WS-Notification service points, or applying topic namespace documents to permanent topic namespaces.

About this task

A WS-Notification service provides the ability to expose some or all of the messaging resources defined on a service integration bus for use by WS-Notification applications.

A JAX-RPC handler list and WS-Security bindings define the parameters and security policy that are used when making outbound Web service invocations, for example monitoring outbound event notification (in response to a subscribe operation) and controlling demand-based publishers (subscribe, pause and resume).

When you create a Version 6.1 WS-Notification service, the wizard configures three service integration bus inbound services for the WS-Notification service, one for each of the three WS-Notification service roles:

- Notification broker
- Subscription manager
- Publisher registration manager

These inbound services are defined on the same service integration bus as the Version 6.1 WS-Notification service, and each of these inbound services refers to the same bus destination.

You can make Web services-specific modifications to the WS-Notification service behavior by modifying the three associated inbound services. You can also configure a custom property that determines the strictness of the syntax checking of topics used under this WS-Notification service.

To modify a WS-Notification service use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. The WS-Notification services [Collection] form is displayed.
2. In the content pane, click the name of a WS-Notification service in the list. The current settings for this WS-Notification service are displayed in the WS-Notification services [Settings] panel.
3. Modify the following general properties:

   Description
   An optional description of the WS-Notification service.

   Enable dynamic topic namespaces?

   Indicates whether dynamic topic namespaces can be used within the WS-Notification service. That is, whether this service allows dynamic topic namespaces to be created at run time. For more information, see Dynamic topic namespace.

   Use this option to tightly control the topic namespaces that are used when connecting to a particular WS-Notification service (for example for security or auditing requirements). If you deselect this option, any applications that connect to the WS-Notification service and request topics from a dynamic topic namespace are stopped from publishing or receiving messages.

   All messages published to a dynamic topic namespace are inserted with the default message reliability setting of `reliable persistent`. If this value is not acceptable, create a permanent topic namespace and manually configure the attribute to the appropriate value.

   Note: The dynamic topic namespaces used on a particular WS-Notification service are backed by a service integration bus topic space that is created automatically when you create the topic namespace. The syntax of topics used within this topic space is internal to the WS-Notification service implementation.

   Requires registration
   Indicates whether publisher applications are required to register with the broker before they can publish notifications.
4. Modify the JAX-RPC handler list and Web services security settings. These settings are applied to the event notifications exchanged with WS-Notification client applications. For more information about handler lists, see “Working with JAX-RPC handlers and clients” on page 1029. For more information about Web services security resources, see Configuring secure transmission of SOAP messages using WS-Security.

   JAX-RPC handler list
   The JAX-RPC handler list that is applied to outbound requests from the WS-Notification service - for example the broker delivering notifications to a consumer.

Outbound security request binding
The security binding to be used with consumer notifications and remote publisher requests sent by this WS-Notification service.

Outbound security response binding
The security binding to be used with remote publisher responses received by this WS-Notification service.

Outbound security configuration
Specifies the details of how security is applied to requests and responses.

5. Modify the additional properties:

WS-Notification service points
Select this link to configure the deployment of WS-Notification service points on one or more servers. For more information, see “Modifying a Version 6.1 WS-Notification service point” on page 1100.

Permanent topic namespaces
Select this link to configure permanent topic namespaces for the WS-Notification service. For more information, see Permanent topic namespace. When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents.

Custom properties
Select this link to configure additional custom properties for this WS-Notification service. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value.

To specify a timeout time for outbound requests sent from this WS-Notification service, set the following custom property:

```
outbound.timeout
```

The value of this property is the timeout time in milliseconds. If the property is not set, a default timeout of 2 minutes is used.

To determine the strictness of the syntax checking of topics used under this WS-Notification service, set the following custom property:

```
com.ibm.ws.sib.wsn.strictTopicChecking
```

Valid values for this property are TRUE and FALSE:

- If the property value is set to TRUE, the topic syntax rules defined in the WS-Topics standard are strictly enforced. Note that there is a performance overhead compared to the default setting, because each character of a topic is validated against a large list of permitted Unicode characters.
- If the property is omitted or set to FALSE, syntax checking only ensures that the basic topic structure is valid, and character checking is relaxed to allow any character exception “*” and “.” as a topic name part.

6. Modify the inbound service settings for the notification broker, subscription manager or publisher registration manager. For more information, see “Modifying an existing inbound service configuration” on page 1011.
7. Apply any changes, then click **OK**. If the processing completes successfully, the list of WS-Notification services is redisplayed. Otherwise, an error message is displayed.
8. Save your changes to the master configuration. You need not restart the server for the changes to fully take effect if **configuration reload** is enabled for the service integration bus.

Deleting WS-Notification services:

Remove all configuration associated with one or more WS-Notification services. A WS-Notification service provides access to service integration bus resources for Web services publish and subscribe clients.

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification service by using the administrative console as described in this task, or by using the deleteWSNService command.

About this task

When you delete a Version 6.1 WS-Notification service, the associated inbound services and inbound ports are also deleted. The associated endpoint listeners are deleted if they were created in the process of creating a WS-Notification service point and are not used by any other configuration object. When you delete a Version 7.0 WS-Notification service, the associated service point provider applications and service points are also deleted. Do not delete these resources manually, because this might leave the associated configuration information in an inconsistent state.

For both types of WS-Notification service, you choose whether or not the associated service integration bus topic spaces are deleted.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services**. A list of all the WS-Notification services is displayed in a WS-Notification services [Collection] form.
3. Select the check box for every WS-Notification service that you want to remove.
4. Click **Delete**. A panel is displayed asking if the service integration bus topic spaces associated with the WS-Notification service (including those associated with all the WS-Notification topic namespaces) should also be deleted.
5. Optional: Select the check box if you want to delete the associated service integration bus topic spaces.

   Note: Deleting a service integration bus topic space causes exception messages to be generated for WS-Notification applications that reference the topic space, as described in Failures as a result of changes in topic space and topic namespace configurations.
6. Click **Delete**. If the processing completes successfully, the list of WS-Notification services is updated. Otherwise, an error message is displayed.
7. Save your changes to the master configuration.

Creating a new Version 7.0 WS-Notification service point:

You can add an additional Version 7.0 WS-Notification service point to an existing Version 7.0 WS-Notification service. A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new Version 7.0 WS-Notification service point by using the administrative console as described in this task, or by using the createWSNServicePoint command.

About this task

You can define any number of WS-Notification service points for a given WS-Notification service. Each service point defined for the same WS-Notification service represents an alternative entry point to the service. Event notifications published to a particular WS-Notification service point are received by all applications connected to any service point of the same WS-Notification service (subject to subscription on the correct topic) regardless of the particular service point to which they are connected. For more information, see Reasons to create multiple WS-Notification service points.

When you create a Version 7.0 WS-Notification service point you select a bus member on which the WS-Notification service point is configured. You also choose the SOAP version that is supported by the service point, and (optionally) apply JAX-WS handler lists to the NotificationBroker, SubscriptionManager or PublisherRegistrationManager that are exposed through this service point.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the Version 7.0 WS-Notification service points configured for this Version 7.0 WS-Notification service.
3. In the content pane, click **New**. The “New WS-Notification service point” wizard is displayed. For more information about the properties that you set with the wizard, see WS-Notification service points [Settings].
4. Step 1: Configure name, description and select a bus member.
   a. Supply a name and (optional) description for the WS-Notification service point. The service point name forms part of the URL used to access the service point.
   b. From the selection list, select the bus member on which the service point is to be configured. On a single server system there is only one bus member in the list.
   c. Click **Next**.
5. Step 2: Define transport settings.
   a. Select the transport settings for the new service point.

      Service point accessed via HTTP proxy
      If the service point is accessed through a proxy, enable the check box, and type the root of the externally visible endpoint address URL for Web services accessed using this endpoint.

      The URL for the proxy is used to populate the WSDL endpoint address fields when publishing WSDL files to a zip file.

      SOAP Version
      Select the version of SOAP that is supported by the service point. This affects the WSDL definition that is exposed by the Web service.

   b. Optional: Select the JAX-WS handler list settings for the new service point.

      NotificationBroker JAX-WS handler list

      The JAX-WS handler list that is applied to inbound requests from an application to the NotificationBroker endpoint of the WS-Notification service point.

      SubscriptionManager JAX-WS handler list

The JAX-WS handler list that is applied to inbound requests from an application to the SubscriptionManager endpoint of the WS-Notification service point.

PublisherRegistrationManager JAX-WS handler list

The JAX-WS handler list that is applied to inbound requests from an application to the PublisherRegistrationManager endpoint of the WS-Notification service point.

6. Click **Finish**. If the processing completes successfully, the list of Version 7.0 WS-Notification service points for this Version 7.0 WS-Notification service is updated to include the new service point. Otherwise, an error message is displayed.
7. Save your changes to the master configuration.

What to do next

To perform advanced configuration tasks for this WS-Notification service point (for example, adding administered subscribers, publishing WSDL files to zip, and configuring the enterprise application associated with this service point), see "Modifying a Version 7.0 WS-Notification service point."

To configure this WS-Notification service point with Web service qualities of service (QoS) such as reliability or security, see "Configuring a Version 7.0 WS-Notification service with Web service QoS" on page 1070.

You can also use the administrative console to work with runtime information for service points. For more information, see "Interacting at run time with WS-Notification" on page 1077.

Modifying a Version 7.0 WS-Notification service point:

Modify the **description**, **SOAP Version**, and **JAX-WS handler list** properties of a Version 7.0 WS-Notification service point, and follow links to complete advanced configuration such as modifying the administered subscribers, applying policy sets to enable WS-ReliableMessaging, publishing the WSDL files to zip files, and configuring the enterprise application that is associated with this service point.

About this task

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the service points configured for this Version 7.0 WS-Notification service.
3. In the content pane, click the name of a Version 7.0 WS-Notification service point in the list. The current settings for this Version 7.0 WS-Notification service point are displayed in the WS-Notification service points [Settings] form.
4. Modify the following general properties:

   Description
   An optional description of the WS-Notification service point.

Associated bus member
The name of the bus member on which this WS-Notification service point is deployed.

SOAP Version

Defines the version of SOAP supported by the service point. This affects the WSDL definition that will be exposed by the web service. Permitted values are “1.1” for SOAP 1.1 (the default), and “1.2” for SOAP 1.2.

NotificationBroker JAX-WS handler list

The JAX-WS handler list that is applied to inbound requests from an application to the NotificationBroker endpoint of the WS-Notification service point.

SubscriptionManager JAX-WS handler list

The JAX-WS handler list that is applied to inbound requests from an application to the SubscriptionManager endpoint of the WS-Notification service point.

PublisherRegistrationManager JAX-WS handler list

The JAX-WS handler list that is applied to inbound requests from an application to the PublisherRegistrationManager endpoint of the WS-Notification service point.

5. Modify the additional properties:

Administered subscribers
An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time. For more information, see “Modifying a WS-Notification administered subscriber” on page 1103.

Custom properties
Select this link to configure additional custom properties for this WS-Notification service point.

Policy set configuration
The policy set configuration associated with this WS-Notification service point. You can configure policy set and binding information for each port relating to this service point. For more information, see “Configuring a Version 7.0 WS-Notification service with Web service QoS” on page 1070.

Publish WSDL files to zip
Publish the WSDL files for this service point to a zip file. For more information, see “Publishing the WSDL files for a WS-Notification application to a ZIP file” on page 1079.

Service point application
The application associated with this WS-Notification service point. For Version 7.0 WS-Notification services, enterprise applications are used to expose the Web services associated with the WS-Notification service.

6. Apply any changes, then click **OK**. If the processing completes successfully, the list of WS-Notification service points for this Version 7.0 WS-Notification service is redisplayed. Otherwise, an error message is displayed.
7. Save your changes to the master configuration.

What to do next

You can also use the administrative console to work with runtime information for service points. For more information, see “Interacting at run time with WS-Notification” on page 1077.

Deleting WS-Notification service points:

Delete one or more WS-Notification service points and the associated resources (Version 7.0 service point provider applications, or Version 6.1 inbound services and endpoint listeners).

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification service point by using the administrative console as described in this task, or by using the deleteWSNServicePoint command.

Do not use this task to delete service points as part of the process of deleting a WS-Notification service. When you delete a WS-Notification service, the associated service points and resources are automatically deleted.

About this task

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. When you delete a WS-Notification service point, the associated resources (Version 7.0 service point provider applications, or Version 6.1 inbound services and endpoint listeners) are automatically deleted. Do not delete these resources manually, because this might leave the associated configuration information in an inconsistent state.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. A list of all the WS-Notification service points is displayed in a WS-Notification service points [Collection] form.
3. Select the check box for every WS-Notification service point that you want to remove.
4. Click **Delete**. If the processing completes successfully, the list of WS-Notification service points for this WS-Notification service is updated. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

Creating a new Version 6.1 WS-Notification service point:

You can add an additional Version 6.1 WS-Notification service point to an existing Version 6.1 WS-Notification service. A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new Version 6.1 WS-Notification service point by using the administrative console as described in this task, or by using the createWSNServicePoint command.

About this task

You can define any number of WS-Notification service points for a given WS-Notification service. Each service point defined for the same WS-Notification service represents an alternative entry point to the service. Event notifications published to a particular WS-Notification service point are received by all applications connected to any service point of the same WS-Notification service (subject to subscription on

the correct topic) regardless of the particular service point to which they are connected. For more information, see Reasons to create multiple WS-Notification service points.

When you create a Version 6.1 WS-Notification service point you select a bus member on which the WS-Notification service point is configured. You allocate a service point to a given bus member by specifying an endpoint listener that is configured for that bus member. You also choose the type of Web service binding (SOAP over HTTP or SOAP over JMS) that is used for the WS-Notification service point.

The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

1. Start the administrative console
2. Navigate to **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the Version 6.1 WS-Notification service points configured for this Version 6.1 WS-Notification service.
3. In the content pane, click **New**. The “New WS-Notification service point” wizard is displayed. For more information about the properties that you set with the wizard, see WS-Notification service points [Settings].
4. Use the wizard to create the new Version 6.1 WS-Notification service point configuration by completing the following steps.
   a. Supply a name and (optional) description for the WS-Notification service point, and from the selection list select the bus member on which the service point is to be configured, then click **Next**. The service point name forms part of the URL used to access the service point (that is, the address of the Web service that is exposed on the chosen server). On a single server system there is only one bus member in the list.
   b. Select a listener application to use to expose the service. Either select an existing endpoint listener for this bus member, or **Create a new endpoint listener**. For more information, see “Creating a new endpoint listener configuration” on page 1019.
5. Click **Finish**. If the processing completes successfully, the list of Version 6.1 WS-Notification service points for this Version 6.1 WS-Notification service is updated to include the new service point. Otherwise, an error message is displayed.
6. Save your changes to the master configuration.

What to do next

You can also use the administrative console to work with runtime information for service points. For more information, see “Interacting at run time with WS-Notification” on page 1077.

Modifying a Version 6.1 WS-Notification service point:

Modify an existing WS-Notification service point. A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service.

About this task

You can define any number of WS-Notification service points for a given WS-Notification service. Each service point defined for the same WS-Notification service represents an alternative entry point to the service. Event notifications published to a particular WS-Notification service point are received by all

applications connected to any service point of the same WS-Notification service (subject to subscription on the correct topic) regardless of the particular service point to which they are connected. For more information, see Reasons to create multiple WS-Notification service points.

The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

To modify a WS-Notification service point use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. The WS-Notification service points [Collection] form is displayed. This form shows all the WS-Notification service points configured for this WS-Notification service.
2. In the content pane, click the name of a WS-Notification service point in the list. The current settings for this WS-Notification service point are displayed in the WS-Notification service points [Settings] form.
3. Modify the general properties. The only general property that you can modify is the **description** property.
4. Modify the additional properties:
   a. Modify the administered subscribers that are associated with this WS-Notification service point. An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time. For more information, see “Modifying a WS-Notification administered subscriber” on page 1103.
   b. Modify the custom properties, if any, that you have set for this WS-Notification service point. These custom properties are name and value pairs that you can use to set internal system configuration properties. In each pair, the name is a property key and the value is a string value.
   a. Modify the inbound ports that are associated with this WS-Notification service point.

      An inbound port describes the Web service enablement of a service destination on a specific endpoint listener, with associated configuration. Each inbound port is associated with an endpoint listener, and you can control which groups of users can access a particular inbound service by making the service available only through specific endpoint listeners. For more information, see Endpoint listeners and inbound ports: Entry points to the service integration bus.

      You can use a JAX-RPC handler list to monitor activity at the port, and take appropriate action (for example logging, or re-routing) depending upon the sender and content of each message that passes through the port. For more information, see Bus-enabled Web services and JAX-RPC handlers.

      You can use WS-Security to set the levels of security to be applied to messages. The security level can be set independently for request and response messages. For more information, see Service integration technologies and WS-Security.

      See also Inbound ports settings.
5. Apply any changes, then click **OK**. If the processing completes successfully, the list of WS-Notification service points for this WS-Notification service is redisplayed. Otherwise, an error message is displayed.
6. Save your changes to the master configuration.

What to do next

You can also use the administrative console to work with runtime information for service points. For more information, see “Interacting at run time with WS-Notification” on page 1077.

Deleting WS-Notification service points:

Delete one or more WS-Notification service points and the associated resources (Version 7.0 service point provider applications, or Version 6.1 inbound services and endpoint listeners).

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification service point by using the administrative console as described in this task, or by using the deleteWSNServicePoint command.

Do not use this task to delete service points as part of the process of deleting a WS-Notification service. When you delete a WS-Notification service, the associated service points and resources are automatically deleted.

About this task

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. When you delete a WS-Notification service point, the associated resources (Version 7.0 service point provider applications, or Version 6.1 inbound services and endpoint listeners) are automatically deleted. Do not delete these resources manually, because this might leave the associated configuration information in an inconsistent state.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points**. A list of all the WS-Notification service points is displayed in a WS-Notification service points [Collection] form.
3. Select the check box for every WS-Notification service point that you want to remove.
4. Click **Delete**. If the processing completes successfully, the list of WS-Notification service points for this WS-Notification service is updated. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

Creating a new WS-Notification administered subscriber:

As part of the configuration of a WS-Notification service point you can configure any number of administered subscribers for that service point. An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new administered subscriber by using the administrative console as described in this task, or by using the createWSNAdministeredSubscriber command.

You should not define an administered subscriber for any of the endpoints exposed by the WS-Notification service on which it is being defined, because this would result in infinite looping of messages through the notification broker.

About this task

An administered subscriber contains the name of a NotificationProducer application or a (different) NotificationBroker endpoint and details of a subscription request (for example topic) that the WS-Notification service point should register as part of the server startup procedure. This allows you to

pre-configure links between the NotificationBroker and a NotificationProducer, which can be a remote NotificationBroker or a NotificationProducer application.

To create a new administered subscriber, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers**. The Administered subscribers [Collection] form is displayed.
2. In the content pane, click **New**.
3. Specify the following properties for this administered subscriber.

   External web service endpoint
   The URL of the external Web service to which the service should subscribe. That is, the endpoint reference (Web address) of a notification producer or notification broker application. For example `http://remoteproducer.com`.

   Dialect
   The dialect in which the topic is expressed. The options are `Simple`, `Concrete`, or `Full`, as defined by the WS-Topics standard.

   Topic The topic on which the service should subscribe. This describes the class of notification messages that are delivered to the WS-Notification service point. For example `stock/IBM`. This property can include wildcards if they are supported by the topic dialect that you select.

   Topic namespace
   The URI that describes the topic namespace in which the specified topic is defined.

   Remote subscription timeout
   The length of time in hours after which the remote subscription will expire if not renewed by the server. This timeout minimizes the potential for orphaned subscriptions in the remote Web service if the local server is uninstalled. Note that this field does not indicate the actual time at which the remote subscription is due to expire. Set the timeout length to something larger than the maximum length of time that the server is expected to remain offline, otherwise the stream of messages from the remote Web service might be interrupted. While the server is running it occasionally renews the remote subscription termination time (with the specified timeout) to prevent it from expiring during normal operation. If not specified, this timeout defaults to 24 (hours).
4. Click **OK**. If the processing completes successfully, the list of administered subscribers associated with the WS-Notification service point is updated to include the new administered subscriber. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

What to do next

You can also use the administrative console to list runtime information for administered subscribers. For more information, see “Listing active WS-Notification administered subscribers” on page 1123.

Modifying a WS-Notification administered subscriber:

As part of the configuration of a WS-Notification service point you can configure any number of administered subscribers for that service point. An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time.

About this task

An administered subscriber contains the name of a NotificationProducer application or a (different) NotificationBroker endpoint and details of a subscription request (for example topic) that the WS-Notification service point should register as part of the server startup procedure. This allows you to pre-configure links between the NotificationBroker and a NotificationProducer, which can be a remote NotificationBroker or a NotificationProducer application.

To modify an administered subscriber, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers**. The Administered subscribers [Collection] form is displayed. This form shows all the administered subscribers configured for this WS-Notification service point.
2. In the content pane, click the name of an administered subscriber in the list. The current settings for this administered subscriber are displayed in the **Configuration** panel.
3. Modify the properties for this administered subscriber.

   External web service endpoint
   The URL of the external Web service to which the service should subscribe. That is, the endpoint reference (Web address) of a notification producer or notification broker application. For example `http://remoteproducer.com`.

   Dialect
   The dialect in which the topic is expressed. The options are `Simple`, `Concrete`, or `Full`, as defined by the WS-Topics standard.

   Topic The topic on which the service should subscribe. This describes the class of notification messages that are delivered to the WS-Notification service point. For example `stock/IBM`. This property can include wildcards if they are supported by the topic dialect that you select.

   Topic namespace
   The URI that describes the topic namespace in which the specified topic is defined.

   Remote subscription timeout
   The length of time in hours after which the remote subscription will expire if not renewed by the server. This timeout minimizes the potential for orphaned subscriptions in the remote Web service if the local server is uninstalled. Note that this field does not indicate the actual time at which the remote subscription is due to expire. Set the timeout length to something larger than the maximum length of time that the server is expected to remain offline, otherwise the stream of messages from the remote Web service might be interrupted. While the server is running it occasionally renews the remote subscription termination time (with the specified timeout) to prevent it from expiring during normal operation. If not specified, this timeout defaults to 24 (hours).
4. Apply any changes, then click **OK**. If the processing completes successfully, the list of administered subscribers associated with the WS-Notification service point is redisplayed. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

What to do next

You can also use the administrative console to list runtime information for administered subscribers. For more information, see “Listing active WS-Notification administered subscribers” on page 1123.

Deleting WS-Notification administered subscribers:

Delete one or more WS-Notification administered subscribers. An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time.

Before you begin

Decide on which method you want to use to configure these resources. You can delete an administered subscriber by using the administrative console as described in this task, or by using the deleteWSNAdministeredSubscriber command.

About this task

To delete one or more administered subscribers, use the administrative console to complete the following steps :

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **Administered subscribers**. A list of all the administered subscribers is displayed in an Administered subscribers [Collection] form.
2. Select the check box for every administered subscriber that you want to remove.
3. Click **Delete**. If the processing completes successfully, the list of administered subscribers for this WS-Notification service is updated. Otherwise, an error message is displayed.
4. Save your changes to the master configuration.

Creating a new WS-Notification permanent topic namespace:

Create a new permanent topic namespace. A topic namespace is a grouping of topics that allows information to be shared between applications. You use a permanent topic namespace to statically define the association between a WS-Notification topic namespace URI and a service integration bus topic space destination.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new WS-Notification permanent topic namespace by using the administrative console as described in this task, or by using the createWSNTopicNamespace command.

You can create many to many relationships between the set of permanent topic namespaces defined in a cell (that is for all WS-Notification services defined in that cell) and the service integration bus topic spaces with which they are associated. These relationships can become quite complex depending upon the topologies required by the applications that connect to the WS-Notification service. For guidance on when certain configurations might or might not be appropriate, see Options for associating a permanent topic namespace with a bus topic space.

About this task

A permanent topic namespace has the following characteristics:

- It enables you to expose an existing service integration bus topic space for use by WS-Notification clients, thus permitting interoperation between the WS-Notification applications and existing publish and subscribe applications connected to the bus such as JMS.
- It allows you to restrict the structure and content of the topic namespace by applying one or more topic namespace documents that describe the required structure.

- It allows the topic namespace to be used as part of a topic space mapping configured on a service integration bus link (between two service integration buses) or a topic mapping as part of a publish and subscribe bridge between a service integration bus and a WebSphere MQ network.

When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents.

You can also set a configuration attribute of a permanent topic namespace to control the reliability (persistence or non persistence) setting that is applied to any messages inserted using a given topic namespace.

To create a new WS-Notification permanent topic namespace, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces**. The Permanent topic namespaces [Collection] form is displayed.
2. In the content pane, click **New**.
3. Specify the following properties for this permanent WS-Notification topic namespace:
   a. Enter a name for the permanent topic namespace. The URI string by which this topic namespace is known. That is, the namespace URI by which WS-Notification applications refer to topics hosted by this namespace. For example `http://widgetproducer.com/prices`.
   b. Associate this new permanent topic namespace with the service integration bus topic space that you want to use to publish and receive messages. From the service integration bus topic space selection list, complete one of the following actions:
      - Choose the name of an existing bus topic space.
      - Choose the offered default name of *this_service_name*`TopicSpace` for a new bus topic space.
      - Choose the option to `Create a new topic space`, then enter a name for the new topic space.
   c. Select from the selection list the service integration bus reliability (quality of service) that is assigned to messages published using this topic namespace. You can choose one of five values. Each value represents one of the service integration bus message reliability levels. The default value is `reliable persistent`, which is the value used by default for JMS Persistent messages.
4. Click **OK**. If the processing completes successfully, the list of permanent WS-Notification topic namespaces is updated to include the new topic namespace. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

What to do next

To view the configuration of the associated service integration bus topic space, see “Showing the properties of a permanent WS-Notification topic namespace.” To apply a topic namespace document, see “Applying a WS-Notification topic namespace document” on page 1108

Showing the properties of a permanent WS-Notification topic namespace:

A topic namespace is a grouping of topics that allows information to be shared between applications.You use a permanent topic namespace to statically define the association between a WS-Notification topic namespace URI and a service integration bus topic space destination.

Before you begin

Decide on which method you want to use to configure these resources. You can show the properties of a permanent WS-Notification topic namespace by using the administrative console as described in this task, or by using the showWSNTopicNamespace command.

About this task

A permanent topic namespace has the following characteristics:

- It enables you to expose an existing service integration bus topic space for use by WS-Notification clients, thus permitting interoperation between the WS-Notification applications and existing publish and subscribe applications connected to the bus such as JMS.
- It allows you to restrict the structure and content of the topic namespace by applying one or more topic namespace documents that describe the required structure.
- It allows the topic namespace to be used as part of a topic space mapping configured on a service integration bus link (between two service integration buses) or a topic mapping as part of a publish and subscribe bridge between a service integration bus and a WebSphere MQ network.

When you create a new WS-Notification permanent topic namespace, you specify the namespace and associate it with one of the service integration bus topic spaces configured on the bus on which the parent WS-Notification service is defined. You cannot modify a permanent topic namespace after it has been created, other than to apply or remove topic namespace documents.

To show the properties of a permanent WS-Notification topic namespace, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces**. The Permanent topic namespaces [Collection] form is displayed. This form shows all the permanent topic namespaces configured for this WS-Notification service.
2. Optional: To view the configuration of the service integration bus topic space that is associated with a given permanent topic namespace, click the link in the **Service integration bus topic space** column. For more information, see Topic space [Settings]
3. Optional: To view the configuration of the namespace documents that are associated with a given permanent topic namespace, click the link in the **Namespace documents** column.

   Note: This link also shows the number of namespace documents (0 or more) that are currently applied.

   The Topic namespace document [Collection] form is displayed. Use this form to apply, show or delete topic namespace documents.

Deleting WS-Notification permanent topic namespaces:

Delete topic namespace definitions from a WS-Notification service. A topic namespace is a grouping of topics that allows information to be shared between applications.You use a permanent topic namespace to statically define the association between a WS-Notification topic namespace URI and a service integration bus topic space destination.

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification permanent topic namespace using the administrative console as described in this task, or you can delete a WS-Notification permanent topic namespace using the wsadmin tool.

About this task

Deleting the topic namespace mapping that was used to establish a (currently active) subscription has the same effect as deleting the underlying service integration bus topic space, and subscriptions that were created using this namespace mapping are deleted. For more information about the effect that deleting a topic namespace has upon new and existing WS-Notification applications, see Failures as a result of changes in topic space and topic namespace configurations.

To delete one or more permanent topic namespaces, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces**. A list of all the permanent topic namespaces for this WS-Notification service is displayed in a Permanent topic namespaces [Collection] form.
2. Select the check box for every permanent topic namespace that you want to remove.
3. Click **Delete**.

   For each selected namespace, if the associated service integration bus topic space was created explicitly by the permanent topic namespace then you are asked whether or not to automatically delete the bus topic space. If the bus topic space was not created by the permanent topic namespace then you are not asked this question and the topic space is not deleted.

   Note: Deleting a service integration bus topic space causes exception messages to be generated for WS-Notification applications that reference the topic space, as described in Failures as a result of changes in topic space and topic namespace configurations.

   If the processing completes successfully, the list of permanent topic namespaces for this WS-Notification service is updated. Otherwise, an error message is displayed.
4. Save your changes to the master configuration.

Applying a WS-Notification topic namespace document:

A topic namespace can optionally have topic namespace documents applied to it that define the structure of the topics that are permitted within the namespace. Use the administrative console to apply a topic namespace document to an existing topic namespace.

Before you begin

This task assumes that you have already created the topic namespace document that you want to apply.

Decide on which method you want to use to configure these resources. You can apply a topic namespace document by using the administrative console as described in this task, or by using the createWSNTopicDocument command.

About this task

For more information about the relationship between a permanent topic namespace and a topic namespace document, see Chapter 5 of the WS-Topics standard.

To apply a topic namespace document to an existing topic namespace, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents**. The Topic

namespace document [Collection] form is displayed. This form shows all the topic namespace documents configured for this permanent topic namespace.

2. In the content pane, click **New**. The Topic namespace document [Settings] form is displayed.
3. Specify the following properties for this topic namespace document:
   a. URL of topic namespace document The URL of the topic namespace document that should be loaded.
   b. Optional: Description An optional description of the topic namespace document.
4. Click **OK**. If the processing completes successfully, the list of topic namespace documents for this permanent topic namespace is updated to include the new document. Otherwise, an error message is displayed.
5. Save your changes to the master configuration.

Showing the contents of a WS-Notification topic namespace document:

A topic namespace can optionally have topic namespace documents applied to it that define the structure of the topics that are permitted within the namespace. Use the administrative console to show the contents of a topic namespace document.

Before you begin

Decide on which method you want to use to configure these resources. You can show the contents of a topic namespace document by using the administrative console as described in this task, or by using the showWSNTopicDocument command.

About this task

For more information about the relationship between a permanent topic namespace and a topic namespace document, see Chapter 5 of the WS-Topics standard.

To show the contents of a WS-Notification topic namespace document, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents**. The Topic namespace document [Collection] form is displayed. This form shows all the topic namespace documents configured for this permanent topic namespace.
2. In the content pane, click the name of a topic namespace document in the list. The contents of this topic namespace document are displayed.

Deleting WS-Notification topic namespace documents:

A topic namespace can optionally have topic namespace documents applied to it that define the structure of the topics that are permitted within the namespace. Use the administrative console to delete one or more WS-Notification topic namespace documents.

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WS-Notification topic namespace document by using the administrative console as described in this task, or by using the deleteWSNTopicDocument command.

About this task

To delete one or more WS-Notification topic namespace documents, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **Permanent topic namespaces** → ***namespace_name*** → **Topic namespace documents**. A list of all the WS-Notification topic namespace documents is displayed in a Topic namespace document [Collection] form.
2. Select the check box for every WS-Notification topic namespace document that you want to remove.
3. Click **Delete**. If the processing completes successfully, the list of WS-Notification topic namespace documents for this WS-Notification topic namespace is updated. Otherwise, an error message is displayed.
4. Save your changes to the master configuration.

Publishing the WSDL files for a WS-Notification application to a ZIP file:

Use the administrative console to download a zip file that contains a WS-Notification application's published WSDL files.

About this task

The ability to publish these WSDL files to a ZIP file is particularly useful in the following circumstances:

- Writing a WS-Notification application that invokes Web service operations against the NotificationBroker application, as described in Writing a WS-Notification application that does not expose a Web service endpoint.
- Viewing the endpoint URLs to which WS-Notification applications connect, by looking in the WSDL file for the NotificationBroker application for Version 7.0 services, or the inbound service for Version 6.1 services.

1. Start the administrative console.
2. Navigate to the "Publish WSDL files to ZIP file [Settings]" form for the WS-Notification application.

   For Version 7.0 WS-Notification services, click one of the following paths:

   - **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **[Additional Properties] Publish WSDL files to zip**
   - **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** → **[Additional Properties] Publish WSDL files to zip**

   For Version 6.1 WS-Notification services, click one of the following paths:

   - **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Related Items] Notification broker inbound service settings** → **[Additional Properties] Publish WSDL files to ZIP file**
   - **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Related Items] Notification broker inbound service settings** → **[Additional Properties] Publish WSDL files to ZIP file**
3. Click on the file name to download a zip file that contains the application's published WSDL files.

Configuring JAX-WS handlers:

A JAX-WS handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. You can create JAX-WS

handlers, chain them together in the form of a handler list, then apply the handler list to a Version 7.0 WS-Notification service point (for inbound invocation handling) or WS-Notification service (for outbound invocation handling).

About this task

The Java API for XML-based Web services (JAX-WS) provides you with a standard way of developing interoperable and portable Web services. To create a JAX-WS handler, you can use a tool such as IBM Rational Application Developer. To enable handlers to perform more complex operations, you chain them together into handler lists. You associate each handler list with one or more Version 7.0 WS-Notification services or service points, so that the handler list can monitor WS-Notification activity and take appropriate action depending upon the sender and content of each inbound or outbound message.

Detailed instructions on how to configure JAX-WS handlers and handler lists for use with Version 7.0 WS-Notification services are provided in the following topics:

- Load JAX-WS handler classes.
- Create a new JAX-WS handler configuration.
- Modify an existing JAX-WS handler configuration.
- Delete JAX-WS handler configurations.
- Create a new JAX-WS handler list.
- Modify an existing JAX-WS handler list.
- Delete JAX-WS handler lists.

Loading JAX-WS handler classes:

A JAX-WS handler interacts with messages through a Version 7.0 WS-Notification service point (for inbound invocation handling) or WS-Notification service (for outbound invocation handling), therefore you need to make the handler class available to the server or cluster that hosts the WS-Notification service point or service that you want to monitor.

Before you begin

This task assumes that you have already created your handler. You can do this using IBM Rational Application Developer or a similar tool.

About this task

Before you can configure a JAX-WS handler for use with WS-Notification, you must make the handler class available to the server or cluster that hosts the WS-Notification service point or service that you want to monitor. To do this, you create a shared library for the class then add the shared library to the class loader for the server.

1. Package the class file for your handler as a JAR file, then copy the JAR file into a convenient directory.

   Make the handler class available to the application server in one of the following ways:

   - Copy the individual class file into a directory structure under *app_server_root*/classes that matches the package name of the class, where *app_server_root* is the root directory for the installation of WebSphere Application Server. For example a handler class com.ibm.jaxws.handler.TestHandler is copied into the *app_server_root*/classes/com/ibm/jaxws/handler directory.
   - Package the class files for all your handlers as a JAR file, then copy it into the *app_server_root*/lib/app directory.
2. Start the administrative console.
3. Create a shared library for the JAR file.
   a. Navigate to **Environment** → **Shared libraries**.

b. Set the scope at which you want the new library to be visible, then click **New**.
c. Give the new library a name.
d. Set the class path to the directory and file name for your handler JAR file.
e. Save your changes to the master configuration.

For more information, see Creating shared libraries.

4. Create a class loader for the server on which you want to make the JAR file available.
   a. Navigate to **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server Infrastructure] Java and Process Management** → **Class loader**.
   b. Click **New**.
   c. Click **OK**.
   d. Save your changes to the master configuration.

   For more information, see Configuring class loaders of a server.
5. Add the shared library to the class loader for the server.
   a. Navigate to **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server Infrastructure] Java and Process Management** → **Class loader** → ***class_loader_name*** → **[Additional Properties] Shared library references**.
   b. Click **Add**.
   c. Click on the name of your new library, then click **OK**.
   d. Save your changes to the master configuration.

   For more information, see Associating shared libraries with servers.

What to do next

You are now ready to create a new JAX-WS handler configuration by using the administrative console or by using the createJAXWSHandler command.

Creating a new JAX-WS handler configuration:

Create a Java API for XML-based Web services (JAX-WS) handler configuration for use, as part of a handler list, with Version 7.0 WS-Notification services.

Before you begin

You can create a new JAX-WS handler configuration by using the administrative console as described in this topic, or by using the createJAXWSHandler command.

This task assumes that you have already created your handler. You can do this using IBM Rational Application Developer or a similar tool. You must also make the handler class available to the server or cluster that hosts the WS-Notification service points (for inbound invocation handling) or WS-Notification services (for outbound invocation handling) that you want to monitor, as detailed in “Loading JAX-WS handler classes” on page 1111.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. To make WebSphere Application Server aware of your handler, and to make the handler available for inclusion in one or more handler lists, you use the administrative console to create a new handler configuration.

1. Start the administrative console.

2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handlers**. The JAX-WS handlers collection form is displayed.
3. Click **New**. The JAX-WS handlers settings form is displayed.
4. Type the following general properties:
   Name Type the name by which the handler is known.

   This name must be unique at cell scope, and it must obey the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   For example `TestHandler`.

   Description
   Type the (optional) description of the handler.

   Class name
   Type the name of the class that is to be instantiated. This name must be a fully qualified java class name. For example `com.ibm.jaxws.handler.TestHandler`.

   Note: You can configure multiple instances of a handler by creating each instance with a different handler name, and pointing to the same handler class.
5. Click **OK**. The general properties for this item are saved, and the additional properties options are made available.
6. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handlers is updated to include the new handler. Otherwise, an error message is displayed.

What to do next

To use this handler, add it to a handler list as described in Creating a new JAX-WS handler list or Modifying an existing JAX-WS handler list.

Modifying an existing JAX-WS handler configuration:

Modify a Java API for XML-based Web services (JAX-WS) handler configuration for a handler that is used, as part of a handler list, with Version 7.0 WS-Notification services.

Before you begin

You can modify a JAX-WS handler configuration by using the administrative console as described in this topic, or by using the modifyJAXWSHandler command.

If you modify a handler class but do not change the class name, you do not need to modify the handler configuration as described in this topic. You just need to stop then restart the servers or clusters that host the services or service points that this handler monitors.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. You can use the administrative console to list existing handler configurations, and to view and modify their configuration details.

1. Start the administrative console.

2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handlers**. A list of handlers is displayed in a JAX-WS handlers collection form.
3. Click the name of a handler in the list. The current JAX-WS handlers settings for this handler are displayed.
4. Modify the following general properties:
   Name Modify the name of the handler.

   This name must be unique at cell scope, and it must obey the following syntax rules:
   - It must not start with "." (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: \ / , # $ @ : ; " * ? < > | = + & % '

   Note: When you change a handler name, the system looks up all objects that refer to it and updates the name.

   Description
   Modify the (optional) description of the handler.

   Class name
   Change the name of the class that is to be instantiated.

   If you change the class name, you must also make the new handler class available to the server or cluster that hosts the WS-Notification service points (for inbound invocation handling) or WS-Notification services (for outbound invocation handling) that you want to monitor, as detailed in "Loading JAX-WS handler classes" on page 1111.

   Note: You can configure multiple instances of a handler by creating each instance with a different handler name, and pointing to the same handler class.
5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handlers is redisplayed. Otherwise, an error message is displayed.

Deleting JAX-WS handler configurations:

Delete the configuration for one or more Java API for XML-based Web services (JAX-WS) handlers that are configured for use, as part of a handler list, with Version 7.0 WS-Notification services.

Before you begin

You can delete a JAX-WS handler configuration by using the administrative console as described in this topic, or by using the deleteJAXWSHandler command.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request.

When you remove a handler that is currently used by one or more Web services on a service integration bus, the system removes the handler from the handler lists for each associated Web service.

You can use the administrative console to remove one or more handler configurations.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handlers**. A list of handlers is displayed in a JAX-WS handlers collection form.

3. Select the check box for every handler that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of handlers is updated. Otherwise, an error message is displayed.

Creating a new JAX-WS handler list:

Create a Java API for XML-based Web services (JAX-WS) handler list for use with Version 7.0 WS-Notification services.

Before you begin

You can create a new JAX-WS handler list by using the administrative console as described in this topic, or by using the createJAXWSHandlerList command.

You can only add previously-configured handlers to a handler list. To configure a handler, see “Creating a new JAX-WS handler configuration” on page 1112.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. To enable handlers to perform more complex operations, you chain them together into handler lists. The approach taken in WebSphere Application Server is to assign handler lists (rather than individual handlers) to WS-Notification service points (for inbound invocation handling) or WS-Notification services (for outbound invocation handling).

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handler Lists**. The JAX-WS handler lists collection form is displayed.
3. Click **New**. The JAX-WS handler lists settings form is displayed.
4. Type the following general properties:
   Name Type the name by which the handler list is known.

   This name must be unique at cell scope, and it must obey the following syntax rules:
   - It must not start with “.” (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

   For example `TestList`.

   Description
   Type the (optional) description of the handler list.

   JAX-WS handlers
   In the JAX-WS handlers pane, complete the following steps:
   a. Select one or more handlers from the list of available JAX-WS handlers, then click **Add** to move the selected handlers into the list of handlers for this JAX-WS handler list.
   b. Select a handler in the list of handlers for this JAX-WS handler list, then click **Up** or **Down** to change the position of the handler within the list.

   Handlers are applied in the sequence in which they appear in the handler list.

   Note: If you click **Reset**, only the **Name** and **Description** fields are reset to their state when the form was first loaded. The two lists of available and assigned handlers are not reset.
5. Click **OK**. The general properties for this item are saved.

6. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handler lists is updated to include the new handler list. Otherwise, an error message is displayed.

What to do next

To apply this handler list to inbound notification requests received at a Version 7.0 WS-Notification point, associate it with the service point as described in “Creating a new Version 7.0 WS-Notification service point” on page 1095 or “Modifying a Version 7.0 WS-Notification service point” on page 1097. To apply this handler list to outbound notification requests sent by your Version 7.0 WS-Notification service, associate the handler list with the service as described in “Creating a new Version 7.0 WS-Notification service” on page 1081 or “Modifying a Version 7.0 WS-Notification service” on page 1085.

For an overview of the end-to-end task of creating JAX-WS handlers, chaining them together in a handler list, then applying the handler list to a Version 7.0 WS-Notification service point or service, see “Applying a JAX-WS handler list to a WS-Notification service” on page 1068.

Modifying an existing JAX-WS handler list:

Modify the configuration details for a Java API for XML-based Web services (JAX-WS) handler list that is configured for use with Version 7.0 WS-Notification services.

Before you begin

You can modify a JAX-WS handler list by using the administrative console as described in this topic, or by using the modifyJAXWSHandlerList command.

You can only add previously-configured handlers to a handler list. To configure a handler, see “Creating a new JAX-WS handler configuration” on page 1112.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request. To enable handlers to perform more complex operations, you chain them together into handler lists. The approach taken in WebSphere Application Server is to assign handler lists (rather than individual handlers) to WS-Notification service points (for inbound invocation handling) or WS-Notification services (for outbound invocation handling).

You can use the administrative console to list existing handler lists, and to view and modify their configuration details.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handler Lists**. A list of all the handler lists is displayed in a JAX-WS handler lists collection form.
3. Click the name of a handler list in the list. The current JAX-WS handler lists settings for this handler are displayed.
4. Modify the following general properties:
   Name Modify the name of the handler list.

   This name must be unique at cell scope, and it must obey the following syntax rules:
   - It must not start with “.” (a period).
   - It must not start or end with a space.
   - It must not contain any of the following characters: `\ / , # $ @ : ; " * ? < > | = + & % '`

When you change a handler list name, the system looks up all objects that refer to it and updates the name.

Description

Modify the (optional) description of the handler list.

JAX-WS handlers

In the JAX-WS handlers pane, complete the following steps:

a. Select one or more previously-configured handlers from either the list of available JAX-WS handlers or the list of handlers for this JAX-WS handler list, then click **Add** or **Remove** to modify the list of handlers for this JAX-WS handler list.
b. Select a handler in the list of handlers for this JAX-WS handler list, then click **Up** or **Down** to change the position of the handler within the list.

Handlers are applied in the sequence in which they appear in the handler list.

Note: If you click **Reset**, only the **Name** and **Description** fields are reset to their state when the form was first loaded. The two lists of available and assigned handlers are not reset.

5. Save your changes to the master configuration.

Results

If the processing completes successfully, the list of handler lists is redisplayed. Otherwise, an error message is displayed.

What to do next

To apply this handler list to inbound notification requests received at a Version 7.0 WS-Notification point, associate it with the service point as described in “Creating a new Version 7.0 WS-Notification service point” on page 1095 or “Modifying a Version 7.0 WS-Notification service point” on page 1097. To apply this handler list to outbound notification requests sent by your Version 7.0 WS-Notification service, associate the handler list with the service as described in “Creating a new Version 7.0 WS-Notification service” on page 1081 or “Modifying a Version 7.0 WS-Notification service” on page 1085.

For an overview of the end-to-end task of creating JAX-WS handlers, chaining them together in a handler list, then applying the handler list to a Version 7.0 WS-Notification service point or service, see “Applying a JAX-WS handler list to a WS-Notification service” on page 1068.

Deleting JAX-WS handler lists:

DeleteJava API for XML-based Web services (JAX-WS) handler lists that are configured for use with Version 7.0 WS-Notification services.

Before you begin

You can delete a JAX-WS handler list by using the administrative console as described in this topic, or by using the deleteJAXWSHandlerList command.

About this task

A Java API for XML-based Web services (JAX-WS) handler is a Java class that performs a range of handling tasks. For example: logging messages, or transforming their contents, or terminating an incoming request.

When you remove a handler list that is currently used by one or more Web services on a service integration bus, the system removes the handler list for each associated Web service.

You can use the administrative console to remove one or more handler list configurations.

1. Start the administrative console.
2. Navigate to **Service integration** → **WS-Notification** → **JAX-WS Handler Lists**. A list of handler lists is displayed in a JAX-WS handler lists collection form.
3. Select the check box for every handler list that you want to remove.
4. Click **Delete**.

Results

If the processing completes successfully, the list of handler lists is updated. Otherwise, an error message is displayed.

Interacting at run time with WS-Notification

View (and in some cases delete) at run time the active items associated with WS-Notification service points.

Before you begin

This task assumes that you have a fully configured and operational WS-Notification service point.

About this task

A WS-Notification service point defines access to a WS-Notification service on a given bus member through a specified Web service binding (for example SOAP over HTTP). Applications use the bus members associated with the WS-Notification service point to connect to the WS-Notification service. The existence of a WS-Notification service point on a bus member implies that a WS-Notification Web service is exposed from that bus member, and causes Web service endpoints for the notification broker, subscription manager and publisher registration manager for this WS-Notification service to be exposed on the bus member with which the service point is associated. WS-Notification applications use these endpoints to interact with the WS-Notification service.

You can use the administrative console to interact at run time with the following active items associated with WS-Notification service points:

Publisher registrations
: A runtime list of existing publisher registrations is provided. This includes, for each publisher registration, the information that was used to create it and when it will terminate.

Pull points
: A runtime list is provided of the pull points that have been created. This includes, for each pull point, its current termination time and a link to the associated subscription.

Subscriptions
: A runtime view is provided for subscriptions that have been created by applications. This includes, for each subscription, the information that was used to create it and an indication of its current state.

Administered subscribers
: A runtime list is provided of the administered subscribers for a given WS-Notification service point. This includes, for each administered subscriber, an indication of its current state; for example whether the subscription was successfully initialized at start time.

For each WS-Notification service point, runtime information is available for subscriptions, registrations, pull points and administered subscribers. **For each WS-Notification service**, runtime information is available - aggregated for all service points for the service - for subscriptions, registrations and pull points. There is no aggregated view for administered subscribers at the WS-Notification service level.

To access and use the runtime information for active items associated with WS-Notification service points, use the administrative console to complete the following steps:

1. Choose between information for a particular service point, or information aggregated for all service points for a particular service, by completing one of the following substeps:
   a. Optional: For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.
   b. Optional: For runtime information aggregated for all service points for a particular service, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations and pull points for this WS-Notification service.
2. Click **Subscriptions** , **Publisher registrations**, **Pull points** or (for a particular service point) **Administered subscribers**.
   - If you click **Subscriptions**, a panel is displayed that lists the durable subscriptions that have been created by a WS-Notification service point in response to “Subscribe” requests from WS-Notification applications. You can view the subscription name (ID), topic and other information associated with the subscription, and you can view messages held on the durable subscription pending delivery. You can also delete subscriptions.

     Note: Both the WS-Notification subscription and publisher registration resources include the concept of scheduled termination, in which the application indicates a period of time after which the resource is destroyed. “Badly-behaved” in this case describes an application that requests an infinite lifetime for a resource and then does not explicitly delete the resource before it goes away (never to return).
   - If you click **Publisher registrations**, a panel is displayed that lists the publisher registrations that are currently in effect on this WS-Notification service or service point (that is, applications that have registered as publishers). You can view the basic properties of the registration record. You can also delete a publisher registration record.
   - If you click **Pull points**, a panel is displayed that lists the pull points that are currently active on this WS-Notification service or service point. You can view basic properties of the pull point such as the subscription with which it is associated and the time at which it is currently set to expire, and you can navigate to the associated subscriptions where appropriate. You can also delete pull points.
   - For a particular service point, if you click **Administered subscribers**, a panel is displayed that lists the administered subscribers that are currently in effect on this WS-Notification service point. You can use this information to see whether a given subscriber has been successfully initialized.

What to do next

For more detailed information about working with individual runtime panels, see the following topics:

- “Listing or deleting active WS-Notification subscriptions.”
- “Listing or deleting active WS-Notification publisher registrations” on page 1121.
- “Listing or deleting active WS-Notification pull points” on page 1122.
- “Listing active WS-Notification administered subscribers” on page 1123.

Listing or deleting active WS-Notification subscriptions:

List the subscriptions that exist at run time for a particular WS-Notification service point, or for all service points for a particular WS-Notification service.

About this task

The runtime panels for WS-Notification subscriptions list the durable subscriptions that have been created by a WS-Notification service point in response to “Subscribe” requests from WS-Notification applications. You can view the subscription name (ID), topic and other information associated with the subscription, and you can view messages held on the durable subscription pending delivery. You can also delete subscriptions. For example, to tidy up after a badly-behaved application.

Note: Both the WS-Notification subscription and publisher registration resources include the concept of scheduled termination, in which the application indicates a period of time after which the resource is destroyed. “Badly-behaved” in this case describes an application that requests an infinite lifetime for a resource and then does not explicitly delete the resource before it goes away (never to return).

A subscription created by a WS-Notification application can contain more than one topic expression, and each of these topic expressions can be in a different namespace, which can result in service integration bus subscriptions being created in multiple topic spaces.

To display a list of WS-Notification subscriptions that exist at run time, use the administrative console to complete the following steps:

1. Choose between information for a particular service point, or information aggregated for all service points for a particular service, by completing one of the following substeps:
   a. Optional: For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.
   b. Optional: For runtime information aggregated for all service points for a particular service, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations and pull points for this WS-Notification service.
2. Click **Subscriptions** The Subscriptions [Collection] form is displayed. This form shows all the currently active subscriptions for this WS-Notification service point or service. This collection form contains the following information about each list item:

   Subscription id
   The unique identifier of the subscription.

   Topics
   An array of topic names. Note that this will normally only contain one element.

   Delivery state
   The current runtime state of the subscription. In normal operation the subscription will show a state of **OK**. Other states are **PAUSED**, indicating that the application has paused the subscription as defined by the WS-Notification standards, or **ERROR**, indicating that the last attempt to deliver an event notification to the notification consumer was not successful. In the error case, further information on the cause of the failure can be found in the SystemOut log.

   Consumer EPR
   The endpoint reference to which event notifications matching the subscription are sent. That is, the endpoint reference to which notification messages are delivered as specified on the subscribe call.

   Creation time
   The time at which the subscription was created.

Termination time
The time at which the subscription will be deleted.

Pull type
Indicates whether the subscription is being used in pull mode. That is, whether this subscription is being accessed by a pull point. If the asynchronous (push) subscription model is being used, this field displays “No”. If a pull point is being used, this field displays “Yes”.

Service integration bus subscriptions
The service integration bus durable subscription or subscriptions that are associated with the WS-Notification subscription. This is a link to the service integration bus durable subscription detail panel for the subscriptions that contain data for this object.

3. Optional: To observe the runtime state of each of the service integration bus durable subscriptions that have been created, click the link in the **SIBus subscriptions** field.

   If only one durable subscription has been created, the bus durable subscription detail panel is displayed. For more information, see “Administering durable subscriptions” on page 1251.

   If multiple durable subscriptions have been created, for example if the WS-Notification subscription is spanned across multiple service integration bus topic spaces, a runtime collection form is displayed that shows the individual bus durable subscriptions. This form shows all the currently active publisher registrations for this WS-Notification service point or service.

   SIBus subscription
   The service integration bus durable subscription ID. This links to the associated bus durable subscription detail panel. For more information, see “Administering durable subscriptions” on page 1251.

   Topics
   The list of topics that are contained within a single service integration bus topic space (and thus durable subscription).

What to do next

To delete one or more WS-Notification subscriptions, and the associated service integration bus durable subscriptions, enable the check box next to each subscription name then click **Delete**.

Listing or deleting active WS-Notification publisher registrations:

List the publisher registrations that exist at run time for a particular WS-Notification service point, or for all service points for a particular WS-Notification service.

About this task

A WS-Notification application can register itself as a publisher in order to check that it is permitted to publish on the specified list of topics, or to initiate the demand based publisher pattern.

The runtime panels for WS-Notification publisher registrations list the publisher registrations that are currently in effect on this WS-Notification service or service point (that is, applications that have registered as publishers). You can view the basic properties of the registration record. You can also delete a publisher registration record. For example, to tidy up after a badly-behaved application.

Note: Both the WS-Notification subscription and publisher registration resources include the concept of scheduled termination, in which the application indicates a period of time after which the resource is destroyed. “Badly-behaved” in this case describes an application that requests an infinite lifetime for a resource and then does not explicitly delete the resource before it goes away (never to return).

To display a list of WS-Notification publisher registrations that exist at run time, use the administrative console to complete the following steps:

1. Choose between information for a particular service point, or information aggregated for all service points for a particular service, by completing one of the following substeps:
   a. Optional: For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.
   b. Optional: For runtime information aggregated for all service points for a particular service, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations and pull points for this WS-Notification service.
2. Click **Publisher registrations** The Publisher registrations [Collection] form is displayed. This form shows all the currently active publisher registrations for this WS-Notification service point or service. This collection form contains the following information about each list item:

   Publisher ID
   The unique identifier of this publisher registration.

   Topic The topic on which the publisher is registered to publish. That is, the list of topics that are contained within a single service integration bus topic space (and thus durable subscription).

   Creation time
   The time at which the registration was created.

   Termination time
   The time at which the registration will be deleted.

   Demand based
   Indicates whether this is a demand based publisher. This displays “Yes” or “No”.

   Producer EPR
   Additional publisher related data. The endpoint reference of the producer that created this registration.

What to do next

To delete one or more publisher registrations, enable the check box next to each registration ID then click **Delete**.

Listing or deleting active WS-Notification pull points:

List the pull points that exist at run time for a particular WS-Notification service point, or for all service points for a particular WS-Notification service.

About this task

To use the synchronous delivery mechanism described by the WS-Notification standards, a WS-Notification application creates a pull point against the notification broker.

The runtime panels for WS-Notification pull points list the pull points that are currently active on this WS-Notification service or service point. You can view basic properties of the pull point such as the subscription with which it is associated and the time at which it is currently set to expire, and you can navigate to the associated subscriptions where appropriate. You can also delete pull points.

To display a list of WS-Notification pull points that exist at run time, use the administrative console to complete the following steps:

1. Choose between information for a particular service point, or information aggregated for all service points for a particular service, by completing one of the following substeps:
   a. Optional: For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.
   b. Optional: For runtime information aggregated for all service points for a particular service, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification services** → ***service_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations and pull points for this WS-Notification service.
2. Click **pull points** The Pull points [Collection] form is displayed. This form shows all the currently active pull points for this WS-Notification service point or service. This collection form contains the following information about each list item:

   Pull point id
   The unique identifier of the pull point.

   Creation time
   The time at which the pull point was created.

   Termination time
   The time at which the pull point will be deleted.

   Subscription id
   The unique identifier of the subscription associated with the pull point. If the pull point has not yet been supplied to a subscription, the text “Not Associated” is displayed.

What to do next

To delete one or more pull points, enable the check box next to each pull point ID then click **Delete**.

Listing active WS-Notification administered subscribers:

List the administered subscribers that exist at run time for a particular WS-Notification service point. An administered subscriber provides a mechanism for the WS-Notification service point to subscribe to an external notification producer at server startup time.

About this task

The runtime panels for WS-Notification administered subscribers list the administered subscribers that are currently in effect on this WS-Notification service point. You can use this information to see whether a given subscriber has been successfully initialized.

Administered Subscribers are defined on individual WS-Notification service points. There is no aggregated view for administered subscribers at the WS-Notification service level.

To display a list of WS-Notification administered subscribers that exist at run time, use the administrative console to complete the following steps:

1. For runtime information for a particular service point, navigate to either **Service integration** → **WS-Notification** → **Services** → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Services] WS-Notification**

services → ***service_name*** → **[Additional Properties] WS-Notification service points** → ***point_name***, then click the **Runtime** tab. A panel is displayed that contains links to runtime information about subscriptions, registrations, pull points and administered subscribers for this WS-Notification service point.

2. Click **Administered subscribers** The Administered subscribers [Collection] form is displayed. This form shows all the currently active administered subscribers for this WS-Notification service point. This collection form contains the following information about each list item:

   Producer EPR
   The endpoint reference of the remote Web service application from which event notifications are received. That is, the endpoint reference (Web address) of a notification producer or notification broker application. For example `http://remoteproducer.com`. You provided this reference when you created the administered subscriber.

   Topic The topic for which event notifications have been requested.

   Subscription reference
   The string form of the endpoint reference that was returned by the remote service as a result of the subscribe, if successfully created.

   State The current runtime state of the administered subscriber. An icon indicator that displays either green or red to indicate whether the administered subscriber is OK or has failed. Hover over this icon to see a description of the current state. In the OK case, it describes the last successful operation - initially the time at which the subscription was contacted (for server startup) and subsequently the time the last message was received for this subscriber. In the failed case, it describes the details of the last failure.

   Subscription timeout
   The length of time in hours after which the remote subscription will expire if not renewed by the server. This timeout minimizes the potential for orphaned subscriptions in the remote Web service if the local server is uninstalled. Note that this field does not indicate the actual time at which the remote subscription is due to expire. Set the timeout length to something larger than the maximum length of time that the server is expected to remain offline, otherwise the stream of messages from the remote Web service might be interrupted. While the server is running it occasionally renews the remote subscription termination time (with the specified timeout) to prevent it from expiring during normal operation. If not specified, this timeout defaults to 24 (hours).

Chapter 6. Service integration

Service integration technologies

Service integration is a technology that provides asynchronous messaging services. This topic explains about the core technologies on which WebSphere Application Server service integration applications are developed and implemented.

About this task

WebSphere Application Server applications invoke asynchronous messaging services using the Java Messaging Service (JMS) application programming interface (API) to interface to a messaging provider. WebSphere Application Server supports a variety of messaging providers, including service integration which is the built-in messaging provider.

Service integration buses

Application servers or clusters of application servers in a WebSphere Application Server cell can cooperate to provide messaging services. A group of servers and/or clusters which cooperate in this way is called a service integration bus. Each of the cooperating servers or clusters is called a bus member. In the simplest case, a service integration bus consists of a single bus member which is one application server.

A WebSphere Application Server application does not need to be running on a service integration bus member to use its messaging services. If necessary, WebSphere Application Server automatically provides a connection to a suitable bus member.

Different service integration buses can, if required, be connected. This allows applications using one bus (the local bus) to send messages to destinations in another bus (a foreign bus). Note, though, that applications cannot receive messages from destinations in a foreign bus.

Messaging engines

Each service integration bus member contains a component called a messaging engine which processes messaging send and receive requests and which can host destinations. To host destinations, the messaging engine includes a message store where, if necessary, it can hold messages until consuming applications are ready to receive them. Each messaging engine also has a file store or a data store where it can hold messages so that they are not lost when the messaging engine fails.

A service integration bus member which is an application server cluster, can contain one or more messaging engines. These can be configured in a variety of ways which provide flexible availability and workload sharing options.

Why use a service integration bus?

- To connect any kind of application to any other kind of application. This is the traditional WebSphere MQ scenario, where different applications written in different languages are running on different operating systems and you want them all to be able to communicate with each other.
- To connect Java EE applications running in application servers. Service integration is a complete JMS v1.1 provider implementation (not just an API but a working messaging system). The JMS provider is a pure Java implementation that runs within the application server's JVM process. (For persistent messaging, WebSphere Application Server also requires a JDBC compliant database such as DB2.) As a result, JMS messaging is built into WebSphere Application Server and is easily available to any Java EE application deployed in WebSphere Application Server.

  For the situation where many WebSphere Application Server applications are communicating, and you also need to communicate with other non-WebSphere Application Server applications, you must still use WebSphere MQ as the messaging system.

Architectures you can use for interoperating WebSphere Application Server and WebSphere MQ are WebSphere MQ as an external JMS messaging provider, WebSphere MQ links, and WebSphere MQ servers.

What are the benefits of service integration bus?

- Secure externalizing of existing applications: You can use the bus to expose existing applications as Web services, for use by any Web service-enabled tool, regardless of the implementation details. This enables applications or Web services deployed on a server deep inside an enterprise to be made available as Web services on the Internet to customers, suppliers, and business partners. Security options mean that this access can be tightly controlled.
- Return on investment: Business partners can reuse an existing process that you make available as a Web service using the bus. This gives great scope for the reuse of existing assets.
- Protocol transformation: The bus provides support for exposing an existing service implemented in one protocol (for example, SOAP/jms), as something entirely different from clients (for example, SOAP/HTTP). This function is invaluable for ensuring smooth interoperability between businesses that may implement varying Web services protocols in their business applications.
- Messaging benefits: The fact that the bus is built on top of the Java Messaging Service (JMS) delivered in WebSphere Application Server means that it is able to expose messaging artifacts, such as queues and topics, as Web services. It also provides advanced options for asynchronous communication, prioritized message delivery, and quality of service (message persistence).

For more information about what is provided by service integration within WebSphere Application Server, see the following topics:

- “Service integration buses”
- “Messaging engines”
- “Managing data stores” on page 1127
- “Bus destinations” on page 1127
- “Mediations” on page 1127
- “Managing messaging with the default messaging provider” on page 1127
- “Using WebSphere MQ links to connect a bus to a WebSphere MQ network” on page 1614
- “Security” on page 1128
- High availability and workload sharing for service integration technologies
- “Enabling Web services through the service integration bus” on page 1002

Service integration buses

Service integration buses manage the messaging resources that are associated with a server using message-based and service-oriented architectures, including those using the Java Message Service (JMS) interfaces. They are not, however, associated with a specific bus or messaging engine.

- Learning about service integration buses
- “Planning a bus topology” on page 1135
- “Configuring buses” on page 1139
- “Operating buses” on page 1193

Messaging engines

These topics provide information about messaging engines, which provide the processing function on a service integration bus.

- Learning about messaging engines
- “Configuring messaging engines” on page 1152
- Operating messaging engines
- “Managing messaging engines with administrative commands” on page 1204
- SIBAdminCommands: Messaging engine administrative commands for the AdminTask object

- Messaging engine troubleshooting tips

Managing data stores

A data store is a message store that uses a relational database. A data store consists of a set of tables that are in the same database schema. It is used by a messaging engine to store operating information in the database, as well as to preserve essential objects that the messaging engine needs for recovery in the event of a failure.

About this task

A data store consists of the set of tables that a messaging engine uses to store persistent data in a database. Refer to “Data store tables” on page 1217 for a list of the tables that comprise a data store. All the tables in a data store are held in the same database schema. You can create multiple data stores in the same database only by using a different schema for each data store.

Bus destinations

This topic is the entry-point into a set of topics about bus destinations within a service integration bus, to which applications can attach as producers, consumers, or both to exchange messages.

- Learning about bus destinations
- “Configuring bus destinations” on page 1159
- “Configuring message points” on page 1248
- “Managing messages on message points” on page 1194
- “Administering durable subscriptions” on page 1251

Mediations

These topics provide information about mediations, which are used to change how messages are handled at destinations on a service integration bus.

- Learning about mediations
- Learning about programming mediations
- Programming mediations
- “Securing mediations” on page 1254
- “Configuring mediations” on page 1256
- “Configuring mediation points” on page 1266
- “Managing mediations with administrative commands” on page 1268
- “Operating mediations at mediation points” on page 1268
- “Administering messages on mediation points” on page 1270
- Message properties support for mediations
- Error handling in mediations
- Mediation thread pool properties

Managing messaging with the default messaging provider

This topic is the entry point into a set of topics about enabling WebSphere Application Server applications to use messaging resources provided by the default messaging provider.

Before you begin

For messaging between application servers, perhaps with some interaction with a WebSphere MQ system, you can use the default messaging provider as described in this topic. To integrate WebSphere Application Server messaging into a predominantly WebSphere MQ network, you can use the WebSphere MQ messaging provider. You can also use a third party messaging provider. To choose the provider that is best

suited to your needs, see Choosing a messaging provider.

About this task

The default messaging provider is installed and runs as part of WebSphere Application Server, and is based on service integration technologies.

The default messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as “unified” or “common” interfaces). This enables applications to use common interfaces for both point-to-point and publish/subscribe messaging. This also enables both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, this approach is recommended for new applications. The domain-specific interfaces are supported for backwards compatibility for applications developed to use domain-specific queue interfaces, as described in section 1.5 of the JMS 1.1 specification.

You can use the WebSphere Application Server administrative console to configure JMS resources for applications, and can manage messages and subscriptions associated with JMS destinations.

WebSphere Application Server Version 5.1 Java EE applications can use messaging resources of the default messaging provider in WebSphere Application Server Version 7.0. This JMS interoperation from WebSphere Application Server Version 5.1 to later versions is enabled and managed by a WebSphere MQ client link created on the WebSphere Application Server Version 7.0 node. This JMS interoperation is only intended as an aid to the migration from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in WebSphere Application Server Version 7.0.

Note: Java EE applications running under WebSphere Application Server Version 7.0 can use messaging resources of Version 5 embedded messaging without any need for a WebSphere MQ client link.

For more information about using the default messaging provider of WebSphere Application Server, see the following topics:

- Learning about the default messaging provider
- “Configuring resources for the default messaging provider” on page 1584
- “Interoperating with a WebSphere MQ network” on page 1612
- Migrating from WebSphere Application Server Version 5 embedded messaging
- “Managing WebSphere Application Server Version 5.1 JMS use of messaging resources in later versions of the product” on page 1646
- “Configuring the messaging engine selection process for JMS applications” on page 1656
- “Managing messages and subscriptions for JMS destinations” on page 1657
- “Using JMS from standalone clients to interoperate with service integration resources” on page 1659
- “Using JMS from a third party application server” on page 1666

Security

This topic is an overview of the tasks for administering messaging security for a service integration bus.

Before you begin

Review the security requirements for the bus. For guidance, see Planning your security requirements.

About this task

Messaging security protects the bus from unauthorized access. By default, messaging security is enabled for the bus. Providing administrative security is also enabled, messaging security enforces a security policy that prevents unauthorized client applications from connecting to the bus, and accessing bus resources.

There may be circumstances when you do not require messaging security, for example on a development system. In this case, you can disable messaging security.

You can customize the security configuration for the bus using the administrative console, or wsadmin scripting commands. The security configuration controls the following aspects of bus security:

- Authorizing groups of users in the user registry to perform selected operations on bus destinations.
- The transport policies that maintain the integrity of messages in transit on the bus.
- The use of global, and multiple custom security domains.
- The integrity of links between messaging engines, foreign buses and databases.

Use the following tasks to administer messaging security:

- "Securing buses" on page 1272
- "Enabling client SSL authentication" on page 1279
- "Adding unique names to the bus authorization policy" on page 1281
- "Administering authorization permissions" on page 1281
- "Administering permitted transports for a bus" on page 1307
- "Securing messages between messaging buses" on page 1311
- "Securing access to a foreign bus" on page 1312
- "Securing links between messaging engines" on page 1313
- "Controlling which foreign buses can link to your bus" on page 1314
- "Securing database access" on page 1314
- "Securing mediations" on page 1254

Auditing the service integration security infrastructure

Security auditing is an important part of the security infrastructure. Security auditing provides a mechanism for auditable events to be tracked and archived while ensuring the integrity of the records.

Before you begin

Before enabling the security auditing subsystem for service integration, you must enable global security in your environment.

About this task

The primary responsibility of the security infrastructure is to prevent unauthorized usage of resources. Security auditing has two primary goals:

- Confirming the effectiveness and integrity of the existing security configuration.
- Identifying areas where improvement to the security configuration might be needed.

Security auditing achieves these goals by providing the infrastructure that allows you to implement your code to capture and store supported security auditable events. All code other than the Java enterprise application code is considered to be trusted. Each time an enterprise application accesses a resource, any internal application server process that has audit points that are added within their code can be recorded as an auditable event. The security auditing subsystem captures the following types of auditable events:

- Authentication
- Authorization
- Principal and Credential Mapping
- Key management
- System management
- Security policy management
- Audit policy management
- Administrative configuration management

- Administrative runtime management
- User registry and identity management
- Password changes
- Delegation

These types of events can be recorded into audit log files. The audit log flies can be signed and encrypted to ensure data integrity. These audit log files can be analyzed to discover breaches over the existing security mechanisms and to discover potential weaknesses in the current security infrastructure. Security event audit records are also useful for providing evidence, accountability, and vulnerability analysis. The security auditing configuration provides four default filters, a default audit service provider, and a default event factory. This following steps describe how to customize your security auditing subsystem. Additional information specific to messaging is included in the step description where appropriate.

1. Enabling the security auditing subsystem

   Security auditing is not performed unless the audit security subsystem has been enabled. Global security must be enabled for the security audit subsystem to function, as no security events occur if global security is not also enabled.

   To allow messaging security events to be audited, audit security must be enabled:

   a. For each bus to be audited, click **Service integration** → **Buses** → ***security_value***, and select the **Enable the auditing service for this bus** check box.

   b. For publish/subscribe messaging, also on each topic space on the bus being audited, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***, and select the **Enable the auditing service for this topic space** check box.

2. Auditor role

   A user with the auditor role is required to enable and configure the security auditing subsystem. It is important to require strict access control for security policy management. The auditor role provides granularity to support separation of the auditing role from the authority of the administrator.

3. Creating security auditing event type filters

   You can configure event type filters to only record a specific subset of auditable event types in your audit logs. Filtering the event types that are recorded makes for simpler analysis of your audit records by ensuring the records to only display what you selected as important for your an environment.

   The audit events that can be configured for messaging are:

   SECURITY_AUTHN
   This event is produced when the identity of a messaging client or messaging engine connecting to a messaging bus is authenticated.

   SECURITY_AUTHZ
   This event is produced when the identity of a messaging client is checked for access authority to a bus or a message queue when sending, directly or by publication, or receiving messages, directly or by subscription.

   SECURITY_AUTHN_TERMINATE
   This event is produced when the connection between a messaging client or messaging engine and a messaging bus is terminated.

   SECURITY_MGMT_CONFIG
   This event is produced when a messaging client changes the contents of a service data object (SDO) repository in an import or remove operation.

   You can create event filters for each permutation of an event and its possible outcomes such as SUCCESS, DENIED, or error conditions of different levels of severity.

   See messaging security events for more information on which messaging security audit events are produced and when they are produced.

4. Configuring audit service providers for security auditing

The audit service provider is used to format the audit data object that is passed to it before outputting the data to a repository.

5. Configuring audit event factories for security auditing

   The audit event factory gathers the data associated with the auditable events and creates an audit data object. The audit data object is then sent to the audit service provider to be formatted and recorded to the repository.

6. Protecting your security audit data It is important for the recorded audit data to be both secured and with the data integrity ensured. To ensure that access to the data is restricted and secure, you can encrypt and sign your audit data.

7. Configuring security audit subsystem failure notifications

   You can enable notifications to generate alerts when the security auditing subsystem experiences a failure. Notifications can be configured to record an alert in the audit logs or can be configured to send an alert through e-mail to a specified list of recipients.

Results

After successfully completing this task, you audit data is recorded for the selected auditable events that were specified in the configuration.

What to do next

After configuring security auditing, you can analyze your audit data for potential weaknesses in the current security infrastructure and to discover security breaches that might have occurred over the existing security mechanisms.

Enabling Web services through the service integration bus

Web services can use the service integration bus to provide a single point of control, access, and validation of Web service requests and allow control of Web services that are available to different groups of Web service users.

About this task

With bus-enabled Web services you can achieve the following goals:

- Create an *inbound service*: Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Create an *outbound service*: Take an externally-hosted Web service, and make it available internally at a bus destination.

Bus-enabled Web services provide a choice of quality of service and message distribution options, along with intelligence in the form of mediations that allow for the rerouting of messages.

To enable Web services through service integration technologies, complete the following steps:

1. Optional: Learn about bus-enabled Web services. Explore the concepts that underly service integration bus-enabled Web services.
2. Plan your bus-enabled Web services installation. Determine the bus-enabled Web services roles that each server is to perform.
3. Ensure that every server that is to play a bus-enabled Web services role is a member of a service integration bus. For more information, see “Configuring the members of a bus” on page 1145.
4. For every server that is to play a bus-enabled Web services role, install and configure a Service Data Objects (SDO) repository on the server.

   Note: For WebSphere Application Server Version 6.0, you also had to manually install a selection of the following applications:

- The service integration technologies resource adapter (used to invoke Web services at outbound ports).
- The bus-enabled Web services application.
- One or more endpoint listener applications.

For later versions of WebSphere Application Server, these applications are installed automatically as and when needed. For example, the endpoint listener application is installed automatically when you configure an endpoint listener.

5. Create a new endpoint listener configuration for each endpoint listener application that you plan to use to receive inbound service requests.
6. Optional: Create an inbound service. An inbound service is a Web interface to a service that is provided internally (that is, a service provided by your own organization and hosted in a location that is directly available through a service integration bus destination). To configure a locally-hosted service as an inbound service, you associate it with a service destination, and with one or more endpoint listeners through which service requests and responses are passed to the service. You can also choose to have the local service made available through one or more UDDI registries.
7. Optional: Create an outbound service. An outbound service is a Web service that is hosted externally, and is made available through a service integration bus. To make an externally-hosted service available through a bus, you first associate it with a service destination, then you configure one or more port destinations (one for each type of binding, for example SOAP over HTTP or SOAP over JMS) through which service requests and responses are passed to the external service. You get the port definitions from the WSDL, but you can choose which ones you want to create.
8. Optional: Apply additional security to your bus-enabled Web services. By default, the bus-enabled Web services configuration works when WebSphere Application Server security is enabled and your service integration buses are secured. However this level of security does not impose any security restrictions on the users of your bus-enabled Web services configuration. To control how your bus-enabled Web services configuration is used by each group of your colleagues or customers, use the bus-enabled Web services additional security features to enable working with password-protected components and servers, with WS-Security and with HTTPS.

What to do next

For further information on specific aspects of bus-enabled Web services, see the following topics:

- Learning about bus-enabled Web services
- “Installing and configuring the SDO repository” on page 1003
- “Configuring Web services for a service integration bus” on page 1008
- “Administering the bus-enabled Web services core resources” on page 1019
- Securing bus-enabled Web services
- “Passing SOAP messages with attachments through the service integration bus” on page 1052
- SIBWebServices command group for the AdminTask object
- Bus-enabled Web services troubleshooting tips
- Tuning bus-enabled Web services

Service integration topologies

This topic provides information about the service integration topologies that you can configure. A service integration topology can range from a single host running two connected applications to a globally-dispersed set of hundreds or thousands of communicating applications running over the bus.

A service integration topology is based on one or more service integration buses that provide a managed communication fabric that supports service integration through synchronous and asynchronous messaging.

A *service integration bus* supports applications using message-based and service-oriented architectures. A bus is a group of one or more interconnected servers that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

A service integration bus provides the following capabilities:

- Any application can exchange messages with any other application by using a *destination* to which one application sends, and from which the other application receives.
- A message-producing application, that is, a *producer*, can produce messages for a destination regardless of which messaging engine the producer uses to connect to the bus.
- A message-consuming application, that is, a *consumer*, can consume messages from a destination (whenever that destination is available) regardless of which messaging engine the consumer uses to connect to the bus.

The service integration bus is sometimes referred to as the *messaging bus* if it is used to provide the messaging system for JMS applications using the default messaging provider.

Many scenarios require a simple bus topology, for example, a single server. If you add multiple servers to a single bus, you increase the number of connection points for applications to use. Servers, however, do not have to be bus members to connect to a bus. In more complex bus topologies, multiple buses are configured, and can be interconnected to form complex networks. An enterprise might deploy multiple interconnected buses for organizational reasons. For example, an enterprise with several independent departments might want separately administered buses in each location.

With bus-enabled Web services you can achieve the following goals:

- Create an *inbound service*: Take an internally-hosted service that is available at a bus destination, and make it available as a Web service.
- Create an *outbound service*: Take an externally-hosted Web service, and make it available internally at a bus destination.

You can change the service integration topology to suit your needs; for example:

- You can add application servers as new bus members. Each new bus member is automatically assigned a messaging engine, with default data source, and a default exception destination. The messaging engines can communicate with, and use resources provided by, all other messaging engines on the bus.
- You can change the configuration of the data source for a messaging engine, for example to use a different JDBC provider.
- You can create new buses and add application servers as members of those buses. Each bus operates as a separate environment, unless connected by a gateway messaging engine.
- You can connect a message-driven bean to consume messages from a destination on a remote cell

Bus topologies

You can connect buses in different ways depending on your requirements. For example, you can link messaging engines to distribute message workload, and to provide availability in the event of system failure.

A topology that consists of a single messaging engine may be adequate for some applications. Deploying more than one messaging engine, and linking them together, provides the following advantages:

- Distributes the messaging workload across multiple servers.
- Positions message processing close to the applications that are using it, and reduces network traffic. For example, if both sending and receiving applications are running in the same server process, it is inefficient to route all the messages that flow between the two applications through a messaging engine running in a remote server.

- Improves availability in the event of system or link failure. For example, your bus topology can remove a single point of failure, and allow store and forward between two servers.
- Provides improved scalability.
- Accommodates firewalls, or other network restrictions that limit the ability of network hosts to connect to a single messaging engine.

Bus topologies that use multiple security domains have additional advantages provided by the presence of bootstrap members. Only servers that are nominated bootstrap members can access the bus. You can define a subset of servers and clusters that can access the bus, and use this information to isolate buses and the applications that use them, and enhance how you manage your bus topologies.

A bus topology can contain links to WebSphere MQ networks. This allows messages to flow between applications connected to a WebSphere MQ queue manager and applications attached to a service integration bus.

The service integration environment backup

It is advisable to back up your service integration system on a regular basis so that you can restore it in the event of an unrecoverable failure. Learn about the issues when taking backups and restoring from a backup.

For service integration, these are the main sections of the system that you can back up:

The configuration files for the system.
The configuration of the system is stored as XML files. To back up or restore these configuration files, use the relevant command as detailed in Backing up and restoring administrative configurations. Any backup or restore of a service integration environment must include a backup or restore of the configuration files.

The data stores that are accessed by the messaging engines.
Taking or restoring a backup of your data stores is optional. Messages are transient in nature, so you might not want to back up or restore the data stores.

- If you do not take a backup of the data stores, and you have modified your current configuration since it was last backed up, and you then restore the configuration backup, be aware that you may lose messages. For example, if you back up the configuration and then create a bus destination, when you restore the configuration backup the destination will no longer exist. Any messages for this destination will be deleted when the server that hosted that messaging engine is restarted.
- If you do take a backup of your data stores, you must also take a backup of the configuration files. You must take the configuration backup at the same time, and restore it at the same time, as the data store backups; taking the backup and restoring at the same time maintains the consistency of the system and reduce the possibility of loss of messages, or duplication of messages from the time of the backup.

To back up a data store, see “Backing up a data store” on page 1216.

The file stores that are accessed by the messaging engines.
Taking or restoring a backup of your file stores is optional. Messages are transient in nature, so you might not want to back up or restore the files. To back up a file store, see “Backing up a file store” on page 1208.

If your environment includes multiple servers, you should backup all the servers at the same time, otherwise messages from the time of the backup might be lost or duplicated. You should also minimize the amount of message traffic flowing through the system to reduce the possibility of lost or duplicate messages.

When you restart a messaging engine after restoring a backup, you should take steps to minimize loss of messages. See Restoring a data store and recovering its messaging engine .

Planning a bus topology

These topics contain information about planning a service integration bus topology. This information is intended primarily for use by solution architects and designers who are in the early stages of planning a bus topology.

About this task

To plan a bus topology, you must consider the following issues:

- How many buses will the topology comprise?
- How will messaging engines and bus destinations be distributed?
- What are the naming conventions for buses, messaging engines, and other service integration bus resources?

Note: Each bus name in a cell must be unique. It is advisable to use unique names for all buses, even when they are in different cells, because you cannot connect two buses with the same name.

You can find more in depth information about your chosen bus topology in the following topics.

Note: Your bus topology might be a combination of the topologies described in these topics.

Planning issues common to all bus topologies

Some planning issues apply to all types of service integration bus topology. This topic guides you through the common decisions you must make about your bus topology.

About this task

Service integration bus configuration

You need to consider the following:

- The volume of messages that the bus will handle. Depending on the volume of messages anticipated, you might need to adjust the **High message threshold** setting for a bus or messaging engine.
- The transport chain to be used for communication between messaging engines. For more information, see Selecting transport options
- Whether bus security is required. When bus security is enabled, access to the bus itself and to all destinations on the bus must be authorized. If you enable bus security, you may also want to define aliases for authenticating messaging engines and mediations accessing the bus. A single version bus (that is, all bus members are at WebSphere Application Server Version 7.0) does not require an authentication alias. If you create a mixed-version bus (that is, the bus contains a bus member at WebSphere Application Server Version 6.x and another bus member at WebSphere Application Server Version 7.0), you need to define an authentication alias for the WebSphere Application Server Version 6.x bus member, to enable it to establish trust with the other bus members.
- The bus name. It is advisable to choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after a bus is created, so if you need to interoperate with WebSphere MQ in the future, it will be much simpler if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.

Bus member configuration

You need to decide which servers to use for bus members that host the messaging engines.

Messaging engine configuration

You might need to adjust the **High message threshold** setting for a messaging engine depending on the volume of messages anticipated.

Message store configuration

Each messaging engine has a single message store, which can be either a file store or a data store. See “Considerations when choosing between a file store and a data store” on page 1204.

You need to consider the sizes of messages. Larger messages will increase the space that a message store requires.

If you use a data store, the default database system for the data store is Apache Derby Version 10.3. However, you might need to use a different system, such as DB2.

You can select different data store topologies depending on your requirements; for more information see “Planning the configuration of a messaging engine to use a data store” on page 1209.

Bus destinations and their configuration

You need to decide on the number and type of destination, mediations, destination routing path, and quality of service for the destination.

For point-to-point messaging, you define bus destinations as queues and for publish/subscribe messaging, you define bus destinations as topic spaces. For point-to-point messaging only, you must select one bus member as the assigned bus member that is to hold messages for the queue. This action automatically defines a queue point for each messaging engine in the assigned bus member.

You can also define alias destinations to provide a level of abstraction between applications and the underlying target bus destinations. Applications interact with the alias destination, so you can change the target bus destination without changing the application

The reliability quality of service for messages on a destination has implications for performance and the amount of space required for a data store. Higher levels of reliability impact on performance and increase the space a data store requires because fewer messages are discarded.

You should decide how you want to use the bus destinations because you can configure a bus destination to prevent producers sending messages to the destination, or consumers receiving messages from the destination.

Application environment

An application attaches as a client to a messaging engine on the bus, either by an in-process call or across a network using a remote client. A remote client can run in either the Java EE application client environment or the Java EE application server environment. Various transport chains can be used. You might decide to provide samples for use with an application.

Application connections

The way that a messaging engine is selected, and the mechanism that an application uses to reach it is configured on a JMS connection factory. You need to decide which messaging engines the applications should connect to and on the transport chain to be used. For more information on connection factories, see “Configuring resources for the default messaging provider” on page 1584 and on transport options, see Selecting transport options.

WebSphere MQ client links

You might want to define WebSphere MQ client links to enable JMS clients developed for WebSphere Application Server Version 5.1 to use messaging resources on the bus. A WebSphere MQ client link is an optional component of a messaging engine that presents a messaging engine, and thereby the bus, as a WebSphere MQ queue manager to which these JMS clients can attach.

Transaction logs

You need to plan where transaction logs will be placed. See the topic about transactional high availability in the related links.

Planning a single-server bus topology

This topic outlines what you should consider when planning a basic bus topology that contains a single server. A single-server bus contains one messaging engine to which all destinations, for example queues and topic spaces, are assigned.

About this task

You need to decide if a single messaging engine is adequate for your needs. You should consider the following factors:

- Number of client connections
- Message throughput rates
- Size of messages

You should also review the planning issues that are common to all topologies.

Planning a multiple-bus topology

Use this task to plan a topology that contains multiple buses.

About this task

In addition to the planning that is common to all topologies, planning issues include:

- The naming of service integration buses; bus names must be unique.
- How buses are to be linked, that is, directly through a service integration bus link, or through an indirect link. For an indirect link, there can be one or more intermediate buses. For more information, see Direct and indirect routing between service integration buses. You also need to decide on the transport chain required, unless you want to use the default basic transport chain.
- Which messaging engines should contain service integration bus links.
- The distribution of destinations on different messaging engines in each bus. You might want to define alias destinations which make a destination available by a different name, on the same bus or a foreign bus. You might also want to define foreign destinations, which enable applications on one bus to directly access a destination on a foreign bus. If you do not define foreign destinations, you can configure destination defaults to be used. You can combine alias and foreign destinations for further flexibility in your topology. For example:

  Use destination defaults in the following scenarios:

  – you have a development environment and want things to work quickly.
  – you have an application in which destination names are received at run time in message body or headers.

  Use foreign destinations in the following scenarios:

  – you want an environment in which everything is statically defined.
  – you need to override destination defaults for a particular (foreign) destination, for example quality of service settings.

  Use an alias destination in the following scenarios:

  – you need to change the name of a destination, or move a destination to another bus, and want existing applications to carry on working regardless.
  – you need multiple names for the same destination.
- If necessary, the mapping between topic spaces on the local bus and topic spaces on foreign buses.

- The security configuration of the topology.
  - If security is enabled, you need to consider whether you have a single version or a mixed version bus. A mixed version bus has a bus member at WebSphere Application Server Version 6.x and another bus member at WebSphere Application Server Version 7.0. In this case, the bus members establish trust using an authentication alias. A single version bus (where bus members are at WebSphere Application Server Version 7.0) does not require you to define an authentication alias to ensure the secure operation of the bus.
  - If buses in different organizations are connected, you must decide whether to secure connections to a foreign bus with a user ID and password, and optionally with SSL authentication.

Planning a topology that includes WebSphere MQ

Use this task to plan a topology that includes WebSphere MQ.

About this task

In addition to the planning that is common to all topologies, planning issues include:

- The distribution of destinations on different messaging engines in each bus. You might want to define alias destinations because applications can use an alias destination to route messages to a target destination in the same bus or to a foreign bus (including across a WebSphere MQ link to a queue provided by WebSphere MQ). You can also use alias destinations to manage situations where the difference in the allowable name length of a bus destination in WebSphere Application Server and the allowable name length of a WebSphere MQ queue could cause a problem.
- You might want to define foreign destinations so that you can specify properties for those destinations. If you do not define either a foreign destination or an alias destination, the properties will have default settings.
- Which messaging engines should contain a WebSphere MQ link. You can have more than one messaging engine with a WebSphere MQ link in a service integration bus. Also, you can have more than one WebSphere MQ link on a single messaging engine. For example, you may decide to have:
  - One WebSphere MQ link engine with only a sender channel and another WebSphere MQ link engine with only a receiver channel.
  - One WebSphere MQ link to communicate with a WebSphere MQ queue manager or queue-sharing group (known as a “gateway queue manager”) in the WebSphere MQ network.

  Although you can have more than one WebSphere MQ link on a single messaging engine, each WebSphere MQ link must connect to a different gateway queue manager.
- The broker profile and associated topic mapping settings.
- The security configuration of the topology.

Enabling or disabling service integration notification events

You can monitor your service integration environment using notification events.

About this task

Service integration notification events allow you to monitor the activity of your service integration configuration using your own system management application. For more details see Service integration notification events.

You can enable or disable service integration notification events at either the bus or the messaging engine level. The following table illustrates the effect of the two setting levels on the enablement of notifications:

Bus-level notification setting	ME-level notification setting	Are notifications enabled
Not set	Not set	Disabled

Bus-level notification setting	ME-level notification setting	Are notifications enabled
Not set	Enabled	Enabled
Not set	Disabled	Disabled
Disabled	Not set	Disabled
Disabled	Enabled	Disabled
Disabled	Disabled	Disabled
Enabled	Not set	Enabled
Enabled	Enabled	Enabled
Enabled	Disabled	Disabled

1. By using the administrative console, open the **Custom properties** page for either the bus or the messaging engine for which you want to set notification events:
   - Click **Service integration** → **Buses** → ***bus_name*** → **[Additional Properties] Custom properties** to display the custom properties for a bus.
   - Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties** to display the custom properties for a messaging engine.
2. Click **New** to create a new custom property.
3. Enter sib.event.notification as the name of the custom property. Set the value of the custom property to either “enabled” or “disabled” as required.
4. Optional: Enter a description for the custom property.
5. Click **OK** and save your changes to the master configuration.

Service integration buses

Service integration buses manage the messaging resources that are associated with a server using message-based and service-oriented architectures, including those using the Java Message Service (JMS) interfaces. They are not, however, associated with a specific bus or messaging engine.

- Learning about service integration buses
- “Planning a bus topology” on page 1135
- “Configuring buses”
- “Operating buses” on page 1193

Configuring buses

You can configure service integration buses in a variety of ways; for example you can create and apply security to a bus and you can then add servers or server clusters to that bus.

Creating a bus

When you create a service integration bus, you add a new bus in the administrative console, and then add one or more servers or server clusters as bus members. Thereafter, you administer the bus, and its constituent bus members, as a single unit.

Before you begin

- Plan your bus topology. For more information, refer to “Planning a bus topology” on page 1135.
- Plan your security strategy. For more information, refer to Planning your security requirements.

About this task

You can create a bus by using the administrative console. Wizards are provided to help you add a secured bus, and add a member to the bus. A messaging engine is created for most types of bus member, and you are prompted to specify a data store. The exception is WebSphere MQ server: since a messaging engine is not created, a message store is not required. Use the administrative console to take the following steps:

1. Add a secured bus or an unsecured bus. For the steps, refer to “Adding buses.”
2. Add a member to the bus. See “Adding a server as a new bus member” on page 1145. This creates a messaging engine with default properties and that uses the default JDBC data source and Apache Derby Version 10.3 JDBC Provider for its data store.
3. Optional: Configure the messaging engine, if required. See “Configuring messaging engine properties” on page 1152.
4. Optional: Configure the data store for the messaging engine, if required. For a messaging engine in an application server, you can change the data store configuration. You could do this, for example, if you want to use DB2 rather than Apache Derby as the database system. For more information about changing the data store configuration, see “Configuring a JDBC data source for a messaging engine” on page 1213. You can also change the file store configuration. Refer to “Modifying file store configuration” on page 1206.
5. Restart the server. The messaging engine starts when the server starts.

Results

A new bus is created.

What to do next

You can now change the configuration of the bus; for example, by adding additional members, by creating bus destinations, and by creating links to WebSphere MQ networks. You can create other buses and, if required, connect them together.

Adding buses

You can add a new service integration bus by using the administrative console. If messaging security is enabled, security settings are configured for the bus by default. You can add a unsecured bus if you disable messaging security.

Adding a secured bus:

This task uses an administrative console wizard to add a new service integration bus that uses a security domain. A bus can use the global security settings, inherit the security settings that exist at cell level, or use a set of customized settings that are unique to the bus, or shared with another resource.

Before you begin

You must decide which type of security domain to use for the bus. For more information, refer to Planning your security requirements and Messaging security and multiple security domains.

About this task

By default, administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, the wizard prompts you to enable it. You must specify the type of user repository, and the administrative security username and password.

Security settings are held in a security domain. If the bus might contain a WebSphere Application Server Version 6.x bus member, you must select the global domain. You can select any type of domain for a bus that contains only WebSphere Application Server Version 7.0 bus members.

1. Log into the administrative console.
2. Click **Service integration** → **Buses**. A list of buses is displayed.
3. In the content pane, click **New**.
4. Type a name for the new bus. It is advisable to choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after a bus is created, so if you need to interoperate with WebSphere MQ in the future, it will be much simpler if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.
5. Ensure that the **Bus security** checkbox is checked.
6. Click **Next**. The Bus Security Configuration wizard is launched.
7. Read the Introduction panel, and click **Next**.
8. If administrative security is disabled, follow the instructions in the wizard to select, and configure the appropriate user repository.
9. Click **Next**. A summary of the administrative security settings for the bus is displayed.
10. Review the summary, and click **Finish**. Administrative security for the cell is now enabled.
11. By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** .
12. Select a security domain for the bus. The domain you specify depends on the versions of the bus members. If the bus may contain a Version 6.x bus member, you must select the global domain. If the bus will only contain Version 7.0 bus members, you can select the cell-level or a custom domain. If you select a custom security domain, you must also follow the instructions to specify a user realm.
13. Review the summary of your choices, and click **Finish**.
14. Save your changes to the master configuration.

Results

Administrative security is enabled, and a new bus is created with your chosen security settings.

What to do next

You can now add bus members to the bus.

Adding an unsecured bus:

This task adds a new unsecured service integration bus.

About this task

By default, bus security is enabled and you add buses with security configured. Use this task if you want to add a new service integration bus without any security settings. If you want to secure the bus at a later date, you can do so by configuring the security settings for the bus.

To add a bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed.
2. In the content pane, click **New**.
3. Type a name for the new bus. It is advisable to choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after a bus is created, so if you need to interoperate with WebSphere MQ in the future, it will be much simpler if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.
4. Clear the **Bus security** checkbox to disable bus security.
5. Click **Next**.

6. Review the summary of the settings for the new bus, then click **Finish**.
7. Save your changes to the master configuration.

Results

Bus security is disabled, and you have added a new unsecured service integration bus.

What to do next

You can now add servers to the bus.

Configuring bus properties

You can configure how many messages the bus can handle, when to discard messages, and which messaging engines the bus can communicate with. You can also specify changes that can be made to the bus that do not require a restart of the messaging engines.

About this task

You configure bus properties after you have created a bus. To configure the properties of a bus, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name***.
2. Specify the following properties for the bus:

   Description
   A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

   Inter-engine transport chain
   The transport chain used for communication between messaging engines in this bus.

   The transport chain must correspond to one of the transport chains defined in the *Messaging engine inbound transports* settings for the server. All servers automatically have a number of transport chains defined to them, and it is also possible to create new transport chains.

   When you specify the name of a transport chain, that chain must be defined to all servers hosting messaging engines in the bus. Otherwise, some messaging engines might not be able to communicate with their peers in the bus.

   When the use of permitted chains is enforced and a protocol is not specified for intra-bus communications then InboundSecureMessaging is assumed instead of InboundBasicMessaging. This can be overridden by setting the protocol attribute in the bus configuration. If InboundSecureMessaging is not a permitted chain then an error occurs.

   Discard messages
   Whether messages on a deleted message point should be retained at a system exception destination or can be discarded. Select this option to indicate that after a queue has been deleted, any messages left in the data store for that queue should be discarded.

   Configuration reload enabled
   Select this option to enable automatic update of configuration information on all the messaging engines on the bus.

   Changes to bus destinations or mediations are applied when destinations or mediations are added to or removed from the bus.

   Changes to the modifiable configuration information for any foreign bus connections are also updated automatically. The time when these changes take effect varies:

Foreign Bus Connection properties
Immediately

WebSphere MQ link properties
On channel restart, except Description (immediately), and Initial State (on messaging engine restart)

MQ sender channel properties
On channel restart, except Initial State (on messaging engine restart or sender channel creation)

MQ receiver channel properties
On channel restart, except Initial State (on messaging engine restart or receiver channel creation)

Publish/subscribe broker profile (0 to n) properties
Immediately

Service integration bus link properties
On link restart, except Description (immediately), and Initial State (on messaging engine restart or link creation)

To ensure that dynamic configuration updates are made on an application server, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] SIB service** then select **Configuration reload enabled**.

To ensure that dynamic configuration updates are made to each node, click **System administration** → **Console Preferences** to display the Console Preferences window then select **Synchronize changes with nodes**.

Default messaging engine high message threshold
A threshold above which the messaging system will take action to limit the addition of more messages to a message point. When a messaging engine is created on the bus, the value of this property is used to set the default high message threshold for the messaging engine.

3. Specify topology, destination resources, Web services and additional properties for the bus, as required. To restrict the number of audit messages, add a custom property called `audit.bus.authentication` with one of the following values:
   - `all` (audit all attempts to authenticate to the bus)
   - `failure` (audit only failure to authenticate to the bus)
   - `none` (do not audit any attempts to authenticate to the bus)
4. Click **OK**.
5. Save your changes to the master configuration.

Listing the buses

You can view the list of the service integration buses that currently exist. You can decide which buses you want to change, for example, to add a server to a bus.

About this task

After you list the service integration buses, you can add or delete a bus. Also, you can select a bus to view its details and perform further actions, such as adding a member to a bus, creating a foreign bus connection, or applying security to a bus. To list the buses, use the administrative console to complete the following step.

In the navigation pane, click **Service integration** → **Buses**.

Results

A list of buses is displayed in the content pane.

What to do next

You can now add or delete buses, or click on a bus name to display or configure its properties.

Displaying the topology of a service integration bus

You can display a tree view of the members of a bus, and the messaging engines used by a selected bus, in the administrative console. You can also view the runtime status for each messaging engine.

About this task

Use this task to display a *local topology* view of the bus members and messaging engines in a service integration bus. You can use this view to add servers as members of the bus.

To display the local topology view of a service integration bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
2. In the content pane, click the name of the bus. For example, `SCA.SYSTEM.localhostCell01.Bus`
3. Click the tab **Local Topology**

Results

The topology of the bus is displayed as an expandable tree.

What to do next

You can expand nodes of the tree to display the bus members and their messaging engines.

You can add servers as members of the bus.

Deleting a bus

You can delete a service integration bus, for example if it is no longer in use.

Before you begin

Note: When you delete a bus, each messaging engine on the bus is also deleted and any associated messages are discarded. If you subsequently create a new bus with the same name as the deleted bus, the new bus will not have the same universal unique identifier (UUID) as the deleted bus.

About this task

To delete a bus, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
2. In the content pane, select the bus that you want to delete.
3. Click **Delete**.
4. Click **OK**.
5. Save your changes to the master configuration.

What to do next

You must disable the SIB Service at server startup.

Configuring the members of a bus

Use the administrative console to add, remove, and list the members of a bus.

Adding a server as a new bus member:

The members of a service integration bus are the application servers within which messaging engines for that bus can run. When you add a new bus member, you configure its message store, which is either a file store or a data store.

About this task

If you add a server as a member of a bus, WebSphere Application Server creates a messaging engine for the server. By default, the messaging engine is configured to use a file store. If you choose a data store, you have the choice of using the default JDBC data source and Derby JDBC Provider for its data store. If you do not want to use the default data source configuration, you can choose to use a different data source or you can configure the data store to use a different JDBC provider. If you subsequently delete a bus member and then recreate it, you should make sure that you understand the life cycle of the file store or a data store. Refer to Data store life cycle and Learning about file stores for details.

You can optionally tune the initial and maximum Java virtual machine (JVM) heap sizes. Tuning the heap sizes helps to ensure that application servers hosting one or more messaging engines are provided with an appropriate amount of memory for the message throughput you require.

If security is enabled, and the bus has mixed version bus members; (that is, a bus member at WebSphere Application Server Version 6.x and another bus member at Version 7.0), the bus members establish trust using an authentication alias. If you add a server as a bus member at WebSphere Application Server Version 6.x, and it is the first bus member at this level, you need to select or create an authentication alias during this task.

To add an application server as a member of a bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bus members**. A list of members in the bus is displayed.
2. Click **Add**.
3. Select scope of the new bus member: this is one of *Server*, *Cluster* or *WebSphere MQ server*. **Server** is selected by default. Only select the **Cluster** scope in WebSphere Application Server environments that support server clusters.
4. Make your selection and click **Next**.
5. Select the type of message store: it is either a file store or a data store. For more information, see File stores and Data stores. **File store** is selected by default.
6. Click **Next**.

   Note: If you use a file store and want to change the default values, you can change them here. For more information refer to “Modifying file store configuration” on page 1206.

   Note: If you use a data store and want the messaging engine in the bus member to use a non-default data source, select **Create new data source** and fill in **Data source JNDI name** field.
7. Click **Next**.

   Note: You can view the current settings of the initial and maximum Java Virtual Machine (JVM) heap sizes. If you want to tune performance by changing the current settings, select the **Change heap sizes** check box and enter the required changes in the **Proposed heap sizes** fields.
8. Click **Next**.

9. If security is enabled, and adding this server creates a mixed version bus, the wizard prompts for an authentication alias. Do one of the following:
   - Select an existing authentication alias.
   - Create a new authentication alias. You need to specify a unique alias name and password.
10. Click **Finish** to confirm the creation of the bus member.
11. Save your changes to the master configuration. You must restart the server for the changes to take effect.

Results

The member is added to the bus and a messaging engine is created for that member.

What to do next

Next, you can configure the messaging engine. For more information about configuring messaging engines and their message stores, see the related tasks.

Adding a WebSphere MQ server as a member of a bus:

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. A WebSphere MQ server bus member is used as a bus member for assigning queue points and mediation points to WebSphere MQ queues.

Before you begin

Get details of the client connection from your WebSphere MQ administrator.

Ensure that the WebSphere MQ server has been configured, that the bus has been defined and that the server is not already a member of the bus.

Decide on which method you want to use to configure these resources. You can add the WebSphere MQ server as a bus member by using the administrative console as described in this task, or by using the addSIBusMember command.

About this task

When you add a WebSphere MQ server to one or more buses, messaging engines on these buses can access queues on the target WebSphere MQ installation. When you make the server a bus member, you can override the server connection settings with settings that are specific to the new bus member. This can be useful in a multiple bus topology.

1. Start the administrative console.
2. Navigate to the list of bus members for the bus to which you are adding the WebSphere MQ server.
   Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bus members**.
3. Click **Add**. The “Add a new bus member” wizard is displayed.
4. Select the WebSphere MQ server to add to the bus:
   a. Select the **WebSphere MQ server** radio button.
   b. From the drop-down list, select the server to add.
   c. Click **Next**.
5. Specify the virtual queue manager name.
   When sending messages to WebSphere MQ, the WebSphere MQ gateway queue manager sees the bus as a remote queue manager. The virtual queue manager name is the name that is passed to WebSphere MQ as the name of this remote queue manager. The default value is the name of the

bus. If this value is not a valid name for a WebSphere MQ queue manager, or if another WebSphere MQ queue manager already exists that has the same name, then replace the default value with another value that is a valid and unique name for a WebSphere MQ queue manager. To be valid, the name must meet the following criteria:

- It must contain between 1 and 48 characters.
- It must conform to the WebSphere MQ queue naming rules (see the WebSphere MQ topic Rules for naming WebSphere MQ objects).

6. Optional: To override the server connection settings, select the **Override WebSphere MQ server connection properties** check box.

   When you select this option, the connection properties for the server are made available so that you can change them to settings that are specific to this bus member. For more information about these connection properties, see WebSphere MQ server bus member [Settings].
7. Optional: If you have changed the server connection settings, you can click **Test connection** to test the connection to the associated WebSphere MQ network.
8. Click **Next**.
9. Click **Finish** to confirm.
10. Save your changes to the master configuration.

What to do next

You are now ready to create a WebSphere MQ queue-type destination for the new bus member.

Listing the members of a bus:

You can display a list of servers that have been added as members of a bus.

About this task

You can list the members of a bus, then you can add, remove, or edit the bus members. To list the members of a bus, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
2. In the content pane, select the bus whose members you want to list.
3. In the content pane, under **Topology**, click **Bus members**. A list of members of the bus is displayed.

Removing a member from a bus:

You can remove an application server from a bus so that it is no longer used to process messages.

Before you begin

Note:

- When you remove a member from a bus, all messaging engines on that bus member are also deleted and their associated messages are discarded. If you add the same bus member again, first you must manually delete the old data source for the messaging engines to ensure that once the new messaging engines are created, they can restart.
- If you remove a member from a bus, and it is the only member for that server, you must also disable the SIB Service at server start up.

About this task

To remove a member from a bus, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
2. In the content pane, select the bus from which you want to remove the member.
3. In the content pane, under **Topology**, click **Bus members**. A list of members in the bus is displayed.
4. Select the bus member that you wish to remove.
5. Click **Remove**.
6. Save your changes to the master configuration.

Results

The member is removed from the bus. All the messaging engines on that bus member and the core group policies that are associated with those messaging engines are deleted.

What to do next

If you have removed the only bus member for a server, you must now disable the SIB Service at server start up.

Disabling the service integration service:

This task describes how to disable service integration bus functionality when the application server starts.

About this task

The SIB Service, which provides messaging capability, is enabled automatically when you add a server to a service integration bus. You can choose to disable the SIB service, for example if you have removed the only bus member for a server. You can use the administrative console to disable the SIB Service, or wsadmin scripting. The following example shows a Jython script for disabling the SIB Service:

```
server = AdminConfig.getid('/Server:server1/')
sibService = AdminConfig.list('SIBService', server)
AdminConfig.modify(sibService, [["enable", "false"]])
```

To use the administrative console, complete the following steps.

1. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] SIB service** to display the SIB service settings pane.
2. Click the Configuration tab to display configuration properties for the service integration service.
3. Clear the Enable service at server startup check box.
4. Save your changes to the master configuration.

Results

You have disabled the SIB Service. Service integration bus functionality is not available when the application server starts up.

Administering bootstrap members for a bus

A service integration bus can have bootstrap members. These are bus members, cell members for which the Service Integration Bus Service is enabled, or nominated cell members that can service requests to bootstrap into the bus, depending on the bootstrap policy configured for the bus.

Before you begin

The bus must exist in a WebSphere Application Server Version 7.0 cell, or a mixed version cell with a Version 7.0 deployment manager.

About this task

You can configure the bootstrap member policy for a bus, nominate, list and delete bootstrap members for a selected bus as follows:

Configuring a bootstrap member policy for a bus:

By configuring a bootstrap member policy for a selected service integration bus, you control which servers respond to bootstrap requests from client applications.

Before you begin

You must ensure that the bus is in a WebSphere Application Server Version 7.0 cell, or in a mixed version cell with a Version 7.0 deployment manager.

About this task

A WebSphere Application Server Version 7.0 bus can use bootstrap members to service connection requests from client applications. In this task, you define a bootstrap member policy for a selected bus that defines which servers can bootstrap into the bus. The default bootstrap member policy is all cell members that have the Service Integration Bus Service enabled. You can restrict bootstrap membership to bus members only, or to bus members, and nominated bootstrap members. These are servers that are not bus members, but have been nominated as bootstrap members.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***bus_name***. The configuration panel for the selected bus is displayed.
3. Choose one of the following bootstrap members policies for the bus:

   All members of the cell with the Service Integration Bus Service enabled
   This is the default option. All the servers in the cell that have the Service Integration Bus Service enabled are bootstrap members.

   Bus members and nominated bootstrap members
   Bootstrap membership is restricted to servers that are bus members, or have been nominated as bootstrap members.

   Bus members only
   Bootstrap membership is restricted to servers that are bus members.
4. Click **Apply**.
5. Save your changes to the master configuration.

Results

Requests from client applications to connect to the bus are serviced by the servers in the cell that meet the criteria set by the bootstrap member policy you have configured for the bus.

What to do next

Use the administrative console to list, nominate or delete bootstrap members for the selected bus.

Listing the bootstrap members for a bus:

Use this task to find out which servers in the cell are available to service requests from clients to connect to a particular service integration bus. The list of available servers is restricted by the bootstrap member policy configured for the bus.

Before you begin

You must ensure that the appropriate bootstrap member policy has been configured for the bus. For more information, refer to “Configuring a bootstrap member policy for a bus” on page 1149.

About this task

In this task, you use the administrative console to list the bootstrap members for a selected bus. The list of bootstrap members depends on which of the following bootstrap member policies is configured for the bus:

All members of the cell with the Service Integration Bus Service enabled
This is the default policy. The list includes all the WebSphere Application ServerVersion 7.0 servers in the cell that have the Service Integration Bus Service enabled.

Bus members and nominated bootstrap members
The list includes all the servers that are bus members, or have been nominated as bootstrap members.

Bus members only
The list includes only servers that are members of the bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bootstrap members**.

Results

A list of the bootstrap members for the selected bus is displayed.

What to do next

You can use the administrative console to nominate or remove bootstrap members, or configure a different bootstrap member policy for the bus.

Nominating bootstrap members for a bus:

You can nominate one or more WebSphere Application Server Version 7.0 servers on Version 7.0 cell nodes as bootstrap members for a selected bus. When the Bus members and nominated bootstrap members policy is configured for the bus, the nominated bootstrap members can service requests from client applications to connect to the bus.

Before you begin

You must ensure that the following requirements are met:

- The bus is in a WebSphere Application Server Version 7.0 cell, or a mixed version cell with a Version 7.0 deployment manager.
- The servers that you want to nominate as bootstrap members must be on Version 7.0 cell nodes.

About this task

In this task, you use an administrative console wizard to configure the **Bus members and nominated bootstrap members** policy, and nominate bootstrap members for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bootstrap members**. The Bootstrap members panel displays all the bus members, servers in the cell that have the Service Integration Bus Service enabled.
3. Select the policy **Bus members and nominated bootstrap members**.
4. Click **Apply**. A list of current bus members, and nominated bootstrap members is displayed.

5. Click **New**. The Nominate a bootstrap server wizard is launched.
6. Type the name of the server that you want to add in the **Name** field. The first match in the list of available servers and clusters is highlighted. Click **Add** to add the named server to the list of servers to add.
7. When you have listed all the servers that you want to add, click **Next**.
8. Review the summary of the servers that you have chosen to add.
9. If you want to change your selections:
   a. Click **Previous** to return to the first step of the wizard.
   b. Make the changes you require, and click **Next**.
10. Click **Finish** to add the new bootstrap members to the local configuration.
11. Save your changes to the master configuration. An updated list of bus members and nominated bootstrap members is displayed.

Results

The nominated servers can service requests from client applications to bootstrap into the bus.

What to do next

You can use the administrative console to perform other bus configuration tasks.

Deleting nominated bootstrap members from a bus:

Deleting a nominated bootstrap member from a selected service integration bus prevents the server from servicing requests from client applications to bootstrap into the bus.

Before you begin

You must configure the **Bus members and nominated bootstrap members** policy for the selected bus. For more information, refer to “Configuring a bootstrap member policy for a bus” on page 1149.

About this task

In this task, you use the administrative console to list all the bootstrap members for a selected bus, and then delete the nominated bootstrap members that you no longer want to use to bootstrap into the bus. Note that you can delete only nominated bootstrap members.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bootstrap members**. All the nominated bootstrap members for the bus are listed.
3. Select the nominated bootstrap members that you no longer want to use to bootstrap into the bus, and click **Delete**. The Bootstrap members panel displays an updated list of bootstrap members.
4. Save your changes to the master configuration.

Results

You cannot use the deleted nominated bootstrap members to bootstrap into the selected bus.

What to do next

You can use the administrative console to perform other bus administrative tasks.

Configuring messaging engines

You can configure messaging engines in a variety of ways. For example, you can create and apply security to a messaging engine, then use this engine to send and receive messages. When you add a server cluster to a service integration bus, at least one messaging engine is created automatically. If you also use messaging engine policy assistance, some configuration properties are set automatically.

Configuring messaging engine properties:

You can configure the properties of a messaging engine in the administrative console. For example you can select whether the messaging engine is started automatically when its associated application server is started, how many messages it can process, and target groups that the engine can join.

About this task

In most cases, you can configure the properties of a messaging engine without interrupting the processing of messages by the messaging engine.

1. Start the administrative console.
2. Navigate to **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name***.
3. Configure the messaging engine properties. For information about the properties that you can configure, see the property descriptions in Messaging engines [Settings]
4. Click **OK**.
5. Save your changes to the master configuration.

Listing the messaging engines in a bus:

You can view the list of existing messaging engines in a bus by using the administrative console. You can decide which messaging engines you want to change, for example which buses they are associated with.

About this task

To list the messaging engines in a bus, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that your messaging engine belongs to.
3. In the content pane, under **Topology**, click **Messaging engines**. The list of messaging engines in the bus is displayed.

Removing a messaging engine from a bus:

You can remove a messaging engine from a service integration bus if you no longer require it to send and receive messages on the bus.

Before you begin

You should be wary of deleting and re-creating messaging engines on bus members for which WS-Notification-administered subscribers have been configured, because in some cases this can leave the remote Web service subscription active (and passing notification messages to the local server) even though there is no longer any record of it. For more information, see the WS-Notification troubleshooting tip Deleting administered subscribers and messaging engines.

1. Stop the messaging engine. You can stop either in **Immediate** or **Force** mode, as described in “Stopping a messaging engine” on page 1202.
2. Use the wsadmin command deleteSIBEngine to delete the messaging engine. All service integration bus links, MQ links, and custom properties that are owned by the engine are deleted.

Note: When you remove a messaging engine, WebSphere Application Server does not delete the data store tables automatically. You must remove them manually, or delete all the rows in all the tables. If you do this, a new messaging engine might fail to start if it attempts to use an orphaned data store. Refer to the documentation for your chosen relational database management system for information about deleting tables.

Alternatively, for Apache Derby, you can delete the database's directory, which is located in *profile_root*/databases/com.ibm.ws.sib, where *profile_root* is the directory in which profile-specific information is stored. However, do this only if the messaging engine is the sole user of the database.

For more information, see Data store life cycle.

Similarly, the file store files are not automatically deleted when you delete the messaging engine. You might want to delete the file store files to reclaim disk space.

Listing the messaging engines defined for a server bus member:

You can display a list of messaging engines defined for a server bus member by using the administrative console. You can decide which messaging engines you want to change, for example, which buses they are associated with.

About this task

To display the list of messaging engines, use the administrative console to complete the following steps:

1. In the navigation pane, click either **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** or **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engines**. A list of messaging engines is displayed in the content pane.
2. Optional: Select one or more messaging engines to work with, for example to change the properties of the messaging engine.

Setting up the data store for a messaging engine:

To set up a data store for a messaging engine, the database administrator must first create the database. You must then configure the data source and configure the messaging engine to use it.

Before you begin

Before you use this task, review the information in "Planning the configuration of a messaging engine to use a data store" on page 1209, and ensure that your database administrator has taken any appropriate action.

About this task

This task summarizes the steps that you must perform to set up the data store for a messaging engine. You need to work with your database administrator to complete step 1 of this task.

1. Ask your database administrator to perform the following steps:
   a. Create the database. Refer to "Creating the database" on page 1210
   b. Create users and schemas. Refer to "Creating users and schemas in the database" on page 1211
   c. Create the data store tables. Refer to "Creating the tables" on page 1212
   d. Ensure that the user has sufficient privilege to allow the messaging engine to operate. For more information about the privileges that required for the selected database, refer to "Database privileges" on page 1218.

2. When the database administrator has completed step 1, perform the following steps:
   a. Obtain a user ID from your database administrator.
   b. Configure a data source. Refer to “Configuring a JDBC data source for a messaging engine” on page 1213.
   c. Configure the messaging engine to use the data source. Refer to “Configuring a messaging engine data store to use a data source.”

Configuring a messaging engine data store to use a data source:

After configuring a data source, you can configure a messaging engine data store to use the data source.

Before you begin

This topic assumes that you have chosen or created a bus and a messaging engine, and that the messaging engine specifies “data store” as its message store type.

You must also have configured a data source, as described in “Setting up the data store for a messaging engine” on page 1153.

About this task

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

Use the WebSphere Application Server administrative console to set the data store configuration parameters.

To configure the messaging engine to use a data source, use the administrative console to complete the following steps:

Service integration → Buses → *bus_name* → [Topology] Messaging engines → *engine_name* → [Additional Properties] Message store

1. In the navigation pane, click **Service integration → Buses → *bus_name* → [Topology] Messaging engines → *engine_name***.
2. Check that the **Message store type** is *Data store*.
3. Click **[Additional Properties] Message store**. The data store configuration detail panel is displayed.
4. Specify the following data store configuration details:

   Data source JMDI name
   Type the JNDI name of the data source that provides access to database that holds the data store.

   Schema name
   Type the name of the database schema that contains the tables used by the data store.

   Note: The schema name is usually the same as the user ID that is declared in the authentication alias. With some databases, for example DB2, you can provide an alternative schema name. For more information about the relationship between users and schema, refer to the documentation for your chosen RDBMS.

   Note: When you configure your messaging engine to use an Informix database, you must specify the schema name in lowercase letters.

   When starting, a messaging engine that uses a data store checks to see if its data store exists. If the user has selected the **Create tables** option in the configuration, whose tick-box is ticked by default, the messaging engine creates the tables in its chosen schema.

The **Schema name** field is optional, and:

- The default schema name is *IBMWSSIB*.
- If you delete the text so that field is blank, the messaging engine takes the user id defined in the authentication alias to be the schema name.
- If you define a schema name explicitly, that schema name is used by the messaging engine.

Authentication alias
Select the authentication alias that enables access to the data source.

Note: When you create a new Network Attached Apache Derby data store, by default you get a blank authentication alias.

Create tables
Select the check box if you want WebSphere Application Server to create the database tables automatically. For more information, refer to “Creating the tables” on page 1212.

Note: The user ID that the data store uses to connect to the data source must have sufficient authority to create the database tables and indexes.

Note: The option for WebSphere Application Server to create the tables is not available with DB2 for z/OS. Refer to “Enabling your database administrator to create the data store tables” on page 1212 if you use DB2 for z/OS.

Number of permanent tables
Permanent table contains persistent objects for the data store.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Number of temporary tables
Temporary tables nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Configuring service integration bus links:

You can configure service integration bus links on messaging engines in a variety of ways. For example, you can start, stop, or remove links.

About this task

When you create a foreign bus connection to connect two service integration buses, a service integration bus link is created automatically. The foreign bus connection contains a routing definition, which is a virtual link, and the service integration bus link is the corresponding physical link on the messaging engine.

Configuring foreign bus connections:

You can configure a bus to connect to, and exchange messages with, other messaging networks. To do this, you must configure a foreign bus connection. A foreign bus connection encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus.

About this task

A foreign bus connection is associated with a service integration bus. The bus with which the foreign bus connection is associated is known as the local bus. The foreign bus connection represents another service integration bus, either in the same cell as the local bus or in a different cell, or it represents a WebSphere MQ queue manager. From the local bus, every other bus is regarded as a foreign bus, even if it is a bus in the same cell.

Messages route to a foreign bus either directly through a link between the local bus and the foreign bus, or indirectly through one or more intermediate buses.

If, when you configure the local bus, you select **Configuration reload enabled**, future changes to the configuration information for any foreign bus connections are updated automatically. The time when these changes take effect varies:

Foreign Bus Connection properties
Immediately

WebSphere MQ link properties
On channel restart, except Description (immediately), and Initial State (on messaging engine restart)

MQ sender channel properties
On channel restart, except Initial State (on messaging engine restart or sender channel creation)

MQ receiver channel properties
On channel restart, except Initial State (on messaging engine restart or receiver channel creation)

Publish/subscribe broker profile (0 to n) properties
Immediately

Service integration bus link properties
On link restart, except Description (immediately), and Initial State (on messaging engine restart or link creation)

To ensure that dynamic configuration updates are made to each node, click **System administration** → **Console Preferences** to display the Console Preferences window, then select **Synchronize changes with nodes**.

Configuring the properties of a service integration bus link:

After establishing a service integration bus link you might want to configure the properties of a service integration bus link such as the name of the service integration bus link, or authentication alias used by foreign bus that the link connects to.

About this task

To configure the properties of a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to configure the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you want to configure.
5. Specify the following properties for the service integration bus link:

Name The name of the service integration bus link. In order to work, the name must be the same as the name of the corresponding service integration bus link configured on the target foreign bus.

Description
An optional description for the service integration bus link, for administrative purposes.

UUID The universal unique identifier assigned by the system to the service integration bus link for administrative purposes.

Foreign messaging engine
The messaging engine on the foreign bus to which this link connects.

Note: This foreign bus name must not be altered after it has been configured. If you alter it, any messaging engines that already hold state information about the link will not be able to use the link unless the foreign bus name is reset to its original value.

Target inbound transport chain
The type of transport chain used for communication with the foreign bus. The transport chain name must be the name of the transport chain as defined on the server on which the target messaging engine is hosted.

Bootstrap endpoints
The comma-separated list of endpoints used to connect to a bootstrap server. This property is set in the same way as the **Provider endpoint** property in the JMS connection factory settings. For more information, see steps 1 and 2 of "Configuring a connection to a non-default bootstrap server" on page 1598. Although this task primarily describes how to configure a JMS connection factory, it is also applicable to setting several bootstrap endpoint values if the remote messaging engine is in a cluster.

Note: Service integration bus links over BootstrapTunneledMessaging and BootstrapTunneledSecureMessaging transport chains only work directly between application server instances. Bus links over TunneledMessaging transport chains do not work if an HTTP server is placed in front of either application server instance.

Authentication alias
The name of the authentication alias, used to authenticate access to the foreign bus. The alias must be known to the foreign bus.

Initial state
Whether the link is started automatically when the messaging engine is started. Until started, the gateway link is unavailable. If this property is set to **Started** the service integration bus link is started when the messaging engine is started.

6. Click **OK**.
7. Save your changes to the master configuration.

Listing the service integration bus links:

You can list all the service integration bus links that are linked to a messaging engine.

About this task

To list the service integration bus links for a messaging engine, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that your messaging engine belongs to.
3. In the content pane, under **Additional properties**, click **Messaging engines**. The list of messaging engines in the bus is displayed.

4. In the content pane, select the messaging engine for which you want to list the service integration bus links.
5. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.

Example

The following combinations of **Status** and **Activity** values are possible:

Status	Activity	Meaning
started	inactive	The service integration bus link is started on the local messaging engine but has no connection to the foreign bus. The service integration bus link is attempting to activate a connection to the foreign bus. The service integration bus link on the foreign bus must also be started to successfully activate of a connection between the buses.
started	active	The service integration bus link is started on the local messaging engine and has an active connection to the foreign bus.
stopped	inactive	The service integration bus link is stopped on the local messaging engine and there is no connection to the foreign bus.
unknown	inactive	An error might have occurred in setting up the link, such that the object that is used to report the current state is not available.

What to do next

You can now add or remove a service integration bus link or select a service integration bus link to start, stop, or configure.

Starting a service integration bus link:

When a service integration bus link has been started, it can be used for communicating with its associated messaging engines.

About this task

To start a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to start the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to start.
5. Click **Start**.

Stopping a service integration bus link:

When a service integration bus link has been stopped, it cannot be used for communication until it is restarted.

About this task

To stop a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine to which the service integration bus link that you wish to stop belongs.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to stop.
5. Click **Stop**.

Removing a service integration bus link:

When a service integration bus link to a messaging engine has been removed, it cannot be used for communication until it is recreated.

Before you begin

Before you remove the service integration bus link you must stop the link.

When you remove a service integration bus link, all traffic using the link needs to be dealt with in a similar way to when you remove a destination. For further details, see “Deleting a non-topic space bus destination” on page 1246. Note that express Quality of Service (QoS) messages are discarded.

About this task

To remove a service integration bus link from a messaging engine, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select which the messaging engine has own association with the service integration bus link that you wish to remove.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to remove.
5. Click **Delete**.
6. Save your changes to the master configuration.

What to do next

When you remove a service integration bus link, all traffic using the link must be diverted by alternative routes or held pending further administrative action. Express QoS messages are discarded.

Configuring bus destinations

Use the following tasks to configure permanent bus destinations on service integration buses.

About this task

The steps involved in configuring a bus destination depend on the intended usage of the destination.

For example, the following figure shows a basic scenario based on an application using a JMS queue for point-to-point messaging.

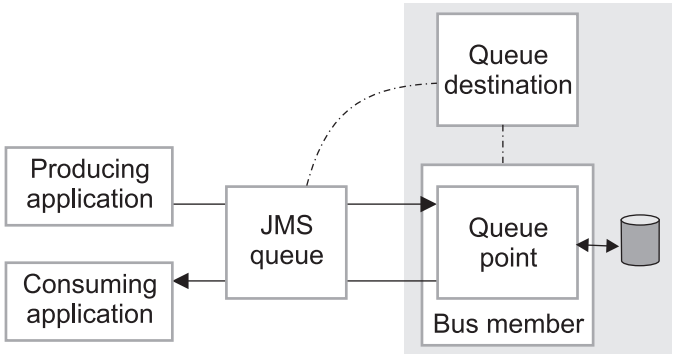


Figure 14. Application use of destinations - basic case

To enable the JMS applications to use a queue destination, you configure the following administrative destination objects:

1. A queue destination on a service integration bus. This configures the properties of the queue, such as the name, and associates the queue with one bus member (an application server). This also automatically configures, on the bus member, a queue point where messages for the queue are held and processed.
2. A JMS queue, which configures the name that applications can use to look up the queue in JNDI. The JMS queue encapsulates the name of the queue destination, as defined in the queue destination above, together with other properties to be used by applications.

After you have created a queue destination, you can optionally configure the queue point to override some properties configured on the queue destination. You can also perform other configuration tasks on the destination and its queue point, and can act on the runtime view.

Each messaging engine has a default exception destination, named `_SYSTEM.Exception.Destinaton.`*`me_name`*. This exception destination can be used to handle messages that cannot be delivered for all bus destination that are localized to the messaging engine. Each bus destination can be configured with a non-default exception destination.

For more information about configuring bus destinations, see the following topics:
- Listing bus destinations
- Creating a bus destination
- Configuring qualities of service for a destination
- Deleting a non-topicspace bus destination
- Deleting a topicspace

Connecting buses

You can connect several service integration buses in a network directly, or indirectly, by creating and configuring foreign bus connections. You can connect a service integration bus and a WebSphere MQ queue manager or queue-sharing group in a similar way. You can enable point-to-point or publish-subscribe messaging across multiple buses.

About this task

For a conceptual overview of connecting different service integration buses, including security issues, refer to Multiple bus topology. You can configure a bus to connect to, and exchange messages with, other messaging networks. To do this, you must configure a foreign bus connection. A foreign bus connection encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus. Messages route to

a foreign bus either directly through a link between the local bus and the foreign bus, or indirectly through one or more intermediate buses.

Configuring foreign bus connections:

You can configure a bus to connect to, and exchange messages with, other messaging networks. To do this, you must configure a foreign bus connection. A foreign bus connection encapsulates information related to the remote messaging network, such as the type of the foreign bus and whether messaging applications are allowed to send messages to the foreign bus.

About this task

A foreign bus connection is associated with a service integration bus. The bus with which the foreign bus connection is associated is known as the local bus. The foreign bus connection represents another service integration bus, either in the same cell as the local bus or in a different cell, or it represents a WebSphere MQ queue manager. From the local bus, every other bus is regarded as a foreign bus, even if it is a bus in the same cell.

Messages route to a foreign bus either directly through a link between the local bus and the foreign bus, or indirectly through one or more intermediate buses.

If, when you configure the local bus, you select **Configuration reload enabled**, future changes to the configuration information for any foreign bus connections are updated automatically. The time when these changes take effect varies:

Foreign Bus Connection properties
: Immediately

WebSphere MQ link properties
: On channel restart, except Description (immediately), and Initial State (on messaging engine restart)

MQ sender channel properties
: On channel restart, except Initial State (on messaging engine restart or sender channel creation)

MQ receiver channel properties
: On channel restart, except Initial State (on messaging engine restart or receiver channel creation)

Publish/subscribe broker profile (0 to n) properties
: Immediately

Service integration bus link properties
: On link restart, except Description (immediately), and Initial State (on messaging engine restart or link creation)

To ensure that dynamic configuration updates are made to each node, click **System administration** → **Console Preferences** to display the Console Preferences window, then select **Synchronize changes with nodes**.

Connecting a bus and a WebSphere MQ gateway queue manager to use point-to-point messaging:

You can connect a service integration bus to a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue sharing group to send or receive messages using point-to-point messaging. One way to do this is to create a foreign bus connection, where the WebSphere MQ queue manager or queue-sharing group is configured as a foreign bus.

Before you begin

To connect a service integration bus and a WebSphere MQ queue manager or queue-sharing group to use point-to-point messaging, the following resources must be defined in WebSphere Application Server:

- A service integration bus that you want to connect from, known as the local bus. The bus must have at least one bus member.

The following resources must be defined in WebSphere MQ:

- A queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network.
- A listener that is configured and running.
- A sender channel (to receive messages on the local bus), a receiver channel (to send messages from the local bus), or both.

About this task

In point-to-point messaging, the sending application specifies the target destination for the message. To receive the message, the receiving application specifies the same destination when it communicates with the messaging provider. Therefore, there is a one-to-one mapping between the sender and receiver of a message.

This task describes one way to achieve point-to-point messaging between WebSphere MQ queue manager or queue-sharing group. For further information about interoperation with a WebSphere MQ network, see the related tasks.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that you want to connect from, that is, the local bus.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**.
4. In the Foreign bus connections pane, click **New** to start the Foreign bus connection wizard.
5. In the Bus connection type pane, ensure that **Direct connection** is selected.
6. In the Foreign bus type pane, select **WebSphere MQ**.
7. In the Local bus details pane, select the messaging engine that you want to use and enter the name of the virtual queue manager, that is, the name by which the virtual queue manager of the service integration bus is known to the WebSphere MQ network.
8. In the WebSphere MQ details pane, enter a name for the foreign bus, that is, the bus that represents the WebSphere MQ queue manager. Enter a name for the WebSphere MQ link that connects to the foreign bus. Ensure that these two names are not the same.
9. Ensure that the **Configure publish-subscribe messaging for this connection** check box is clear.
10. To send messages from the local bus to the WebSphere MQ queue manager, complete the following details:
    a. Ensure that **Enable Service integration bus to WebSphere MQ message flow** is selected.
    b. Enter the WebSphere MQ receiver channel name, host name and communication port.
    c. If the WebSphere MQ queue manager requires a secure connection, select the **Is the WebSphere MQ receiver channel secure?** check box. When this option is selected, the WebSphere MQ receiver channel accepts only connections that have secure sockets layer (SSL) based encryption. The connection is successful only if a set of suitably compatible SSL credentials are associated with the service integration bus outbound channel and the WebSphere MQ receiver channel that it connects to.
11. To receive messages on the local bus from the WebSphere MQ queue manager, complete the following details:
    a. Ensure that **Enable WebSphere MQ to Service integration bus message flow** is selected.

b. Enter the WebSphere MQ sender channel name.

c. Optionally, enter the service integration bus inbound user ID. When the local bus is secure, the inbound user ID replaces the user ID in messages from the foreign bus that arrive at the local bus and is used to authorize whether those messages can access their destinations. Specify an inbound user ID for the local service integration bus under the following circumstances:

   - The foreign bus is in a different security domain, so user IDs in the foreign bus are not recognized in the local bus.
   - You want local control of access to inbound messages to the local bus.

   If the local bus is not secure, the inbound user ID has no effect on messages. If the local bus is secure, the foreign bus is not secure, and an inbound user ID is not set, an inbound message from the foreign bus is only authorized to destinations that allow unauthenticated users access.

12. When the Foreign bus connection wizard is finished, save your changes to the master configuration.

Results

You have created a connection between a service integration bus and a WebSphere MQ queue manager to use point-to-point messaging. You have created a direct foreign bus connection, which contains a routing definition, or virtual link. The physical link, a WebSphere MQ link on the messaging engine for the local bus, is created automatically.

What to do next

You can test the connection.

Connecting a bus and a WebSphere MQ network to use publish-subscribe messaging:

You can connect a service integration bus and a WebSphere MQ network to send and receive messages using publish-subscribe messaging. To do this, you create a foreign bus connection, where the WebSphere MQ network is viewed as a foreign bus.

Before you begin

To connect a service integration bus and a WebSphere MQ network to use publish-subscribe messaging, the following resources must be defined in WebSphere Application Server:

- A service integration bus that you want to connect from, known as the local bus. The bus must have at least one bus member.

The following resources must be defined in WebSphere MQ:

- A queue manager or (for WebSphere MQ for z/OS) queue-sharing group, which acts as the gateway to the WebSphere MQ network.
- A listener that is configured and running.
- A topic and input queue for broker publish-subscribe flow configured in WebSphere MQ.
- A sender channel (to receive messages on the local bus), a receiver channel (to send messages from the local bus), or both.

About this task

In publish-subscribe messaging, the sending application publishes messages to an intermediate broker destination. Multiple receiving applications can subscribe to this destination to receive a copy of any messages that are published. When a message arrives at a destination, the messaging provider distributes a copy of the message to all the receiving applications that subscribe to the destination. There can be a one-to-many relationship between the sender and receiver of a message, depending on how many receiving applications are subscribed to a destination when a message arrives.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that you want to connect from, that is, the local bus.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**.
4. In the Foreign bus connections pane, click **New** to start the Foreign bus connection wizard.
5. In the Bus connection type pane, ensure that **Direct connection** is selected.
6. In the Foreign bus type pane, select **WebSphere MQ**.
7. In the Local bus details pane, select the messaging engine that you want to use and enter the name of the virtual queue manager, that is, the name by which the virtual queue manager of the service integration bus is known to the WebSphere MQ network.
8. In the WebSphere MQ details pane, complete the following details:
   a. Enter a name for the foreign bus, that is, the bus that represents the WebSphere MQ network.
   b. Enter a name for the WebSphere MQ link that connects to the foreign bus. Ensure that the Foreign bus name and MQ link name are different.
   c. Select the **Configure publish-subscribe messaging for this connection** check box.
9. To send messages from the local bus to the WebSphere MQ network, complete the following details:
   a. Ensure that **Enable Service integration bus to WebSphere MQ message flow** is selected.
   b. Enter the WebSphere MQ receiver channel name, host name and communication port.
   c. If the WebSphere MQ gateway queue manager or queue-sharing group requires a secure connection, select the **Is the WebSphere MQ receiver channel secure?** check box. When this option is selected, the WebSphere MQ receiver channel accepts only connections that have secure sockets layer (SSL) based encryption. The connection is successful only if a set of suitably compatible SSL credentials are associated with the service integration bus outbound channel and the WebSphere MQ receiver channel that it connects to.
10. To receive messages on the local bus from the WebSphere MQ network, complete the following details:
    a. Ensure that **Enable WebSphere MQ to Service integration bus message flow** is selected.
    b. Enter the WebSphere MQ sender channel name.
    c. Optionally, enter the service integration bus inbound user ID. When the local bus is secure, the inbound user ID replaces the user ID in messages from the foreign bus that arrive at the local bus and is used to authorize whether those messages can access their destinations. Specify an inbound user ID for the local service integration bus under the following circumstances:
       - The foreign bus is in a different security domain, so user IDs in the foreign bus are not recognized in the local bus.
       - You want local control of access to inbound messages to the local bus.

       If the local bus is not secure, the inbound user ID has no effect on messages. If the local bus is secure, the foreign bus is not secure, and an inbound user ID is not set, an inbound message from the foreign bus is only authorized to destinations that allow unauthenticated users access.
11. From the Publish-subscribe details pane, repeat the following steps for each topic mapping you want to create:
    a. Enter the name of the topic on the local bus.
    b. Select the name of the topic space on the local bus that will map to the topic space on the foreign bus.
    c. Enter the name of the gateway queue manager or queue sharing group for the WebSphere MQ broker configured for broker publish-subscribe flow.
    d. To send messages from the local bus to the WebSphere MQ gateway queue manager or queue-sharing group, enter the name of the queue for the WebSphere MQ broker destination.

e. To receive messages on the local bus from the WebSphere MQ gateway queue manager or queue-sharing group, enter the name of the subscription point that will receive messages.
f. Select the direction of message flow for the publish-subscribe topic mapping. The options available depend on whether you completed details on the WebSphere MQ details pane to send messages, receive messages, or both, on the local bus.
g. Click **Add**.

12. When the Foreign bus connection wizard is finished, save your changes to the master configuration.

Results

You have created a connection between a service integration bus and a WebSphere MQ network to use publish-subscribe messaging. You have created a direct foreign bus connection, which contains a routing definition, or virtual link. The physical link, a WebSphere MQ link on the messaging engine for the local bus, is created automatically.

What to do next

You can test the connection.

Connecting service integration buses to use point-to-point messaging:

You can connect a service integration bus to another service integration bus to send and receive messages using point-to-point messaging. To do this, you create a foreign bus connection.

Before you begin

To connect a service integration bus to another service integration bus to use point-to-point messaging, the following resources must be defined:

- A service integration bus that you want to connect from, known as the local bus. The bus must have at least one bus member.
- A service integration bus that you want to connect to, known as the foreign bus. The bus must have at least one bus member.
- Optionally, to configure a secure connection, an authentication alias.

The buses that you connect must have unique names, because the connection will fail if the buses have the same name.

About this task

In point-to-point messaging, the sending application specifies the target destination for the message. To receive the message, the receiving application specifies the same destination when it communicates with the messaging provider. Therefore, there is a one-to-one mapping between the sender and receiver of a message.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that you want to connect from, that is, the local bus.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**.
4. In the Foreign bus connections pane, click **New** to start the Foreign bus connection wizard.
5. In the Bus connection type pane, ensure that **Direct connection** is selected.
6. In the Foreign bus type pane, ensure that **Service integration bus** is selected.
7. In the Local bus details pane, select the messaging engine that you want to use from the drop-down list.

8. Optionally, enter a name for the inbound user ID. When the local bus is secure, the inbound user ID replaces the user ID in messages from the foreign bus that arrive at the local bus and is used to authorize whether those messages can access their destinations. Specify an inbound user ID for the local service integration bus under the following circumstances:
   - The foreign bus is in a different security domain, so user IDs in the foreign bus are not recognized in the local bus.
   - You want local control of access to inbound messages to the local bus.

   If the local bus is not secure, the inbound user ID has no effect on messages. If the local bus is secure, the foreign bus is not secure, and an inbound user ID is not set, an inbound message from the foreign bus is only authorized to destinations that allow unauthenticated users access.
9. In the Foreign bus details pane, complete the details as appropriate:
   - If the service integration bus you want to connect to is in a different cell from the local bus, complete the following details:
     a. Ensure that **Configure a foreign bus in a remote cell** is selected.
     b. Enter the name of the service integration bus to connect to, that is, the foreign bus. Enter the exact name of the existing service integration bus.
     c. Enter the name of the gateway messaging engine in the foreign bus, that is, the messaging engine to connect to in the foreign bus.
     d. Ensure that the **Configure publish-subscribe messaging for this connection** check box is clear.
     e. Enter the name of the service integration bus link.
     f. Enter one or more bootstrap endpoints, that is, the host, port location, and transport chain for the messaging engine on the foreign bus that the local service integration bus connects to. The port is the SIB_ENDPOINT_ADDRESS (or SIB_ENDPOINT_SECURE_ADDRESS if security is enabled) of the messaging engine. Use the format *hostName*:*portNumber*:*chainName*, separating each bootstrap endpoint by a comma. For more information, see steps 1 and 2 of “Configuring a connection to a non-default bootstrap server” on page 1598. Although this task primarily describes how to configure a JMS connection factory, it is also applicable to setting several bootstrap endpoint values if the remote messaging engine is in a cluster.
   - If the service integration bus you want to connect to is in the same cell as the local bus, complete the following details:
     a. Ensure that **Configure a foreign bus in a local cell** is selected.
     b. Select the name of the service integration bus to connect to, that is, the foreign bus.
     c. Select the name of the gateway messaging engine in the foreign bus, that is, the messaging engine to connect to in the foreign bus.
     d. Ensure that the **Configure publish-subscribe messaging for this connection** check box is clear.
     e. Enter the name of the service integration bus link.
10. Optionally, to secure the connection, in the Foreign bus details pane, complete the following details:
    a. Select the **Secure connection** check box.
    b. Select the type of transport chain to use to communicate with the messaging engine in the foreign bus. Select one of the following:
       - InboundBasicMessaging. InboundBasicMessaging is a predefined transport chain where communication uses the TCP protocol.
       - InboundSecureMessaging. InboundSecureMessaging is a predefined transport chain where communication is secured by using the secure sockets layer (SSL) based encryption protocol over a TCP network. For successful connection, a set of suitably compatible SSL credentials must be associated with the local bus inbound channel and the foreign bus outbound channel.
       - Other, please specify. Select this option to specify your own transport chain and enter the details in the field that appears.

c. Select the name of the authentication alias to use to authenticate access to the foreign bus. The alias must be known to the foreign bus.

11. When the Foreign bus connection wizard is finished, save your changes to the master configuration.

Results

You have created a connection from a local service integration bus to a foreign service integration bus to use point-to-point messaging. You have created a direct foreign bus connection, which contains a routing definition, or virtual link. The physical link, a service integration bus link on the messaging engine for the local bus, is created automatically.

What to do next

You need to create a connection in the opposite direction between the two buses. To do this, repeat the procedure, using the bus you have just connected to as the local bus, and the bus you have just connected from as the foreign bus. Ensure that you use exactly the same name for the service integration bus link. After you create a foreign bus connection for each service integration bus, you can test the connection.

Connecting service integration buses to use publish-subscribe messaging:

You can connect a service integration bus to another service integration bus to send and receive messages using publish-subscribe messaging. To do this, you create a foreign bus connection.

Before you begin

To connect a service integration bus to another service integration bus to use publish-subscribe messaging, the following resources must be defined:

- A service integration bus that you want to connect from, known as the local bus. The bus must have at least one bus member.
- A service integration bus that you want to connect to, known as the foreign bus. The bus must have at least one bus member.
- A topic space on both service integration buses. If the foreign bus is in a remote cell, you need to know the topic space name.
- Optionally, to configure a secure connection, an authentication alias.

The buses that you connect must have unique names, because the connection will fail if the buses have the same name.

About this task

In publish-subscribe messaging, the sending application publishes messages to an intermediate broker destination. Multiple receiving applications can subscribe to this destination to receive a copy of any messages that are published. When a message arrives at a destination, the messaging provider distributes a copy of the message to all the receiving applications that subscribe to the destination. There can be a one-to-many relationship between the sender and receiver of a message, depending on how many receiving applications are subscribed to a destination when a message arrives.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that you want to connect from, that is, the local bus.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**.
4. In the Foreign bus connections pane, click **New** to start the Foreign bus connection wizard.
5. In the Bus connection type pane, ensure that **Direct connection** is selected.

6. In the Foreign bus type pane, ensure that **Service integration bus** is selected.
7. In the Local bus details pane, select the messaging engine that you want to use from the drop-down list.
8. Optionally, enter a name for the inbound user ID. When the local bus is secure, the inbound user ID replaces the user ID in messages from the foreign bus that arrive at the local bus and is used to authorize whether those messages can access their destinations. Specify an inbound user ID for the local service integration bus under the following circumstances:
   - The foreign bus is in a different security domain, so user IDs in the foreign bus are not recognized in the local bus.
   - You want local control of access to inbound messages to the local bus.

   If the local bus is not secure, the inbound user ID has no effect on messages. If the local bus is secure, the foreign bus is not secure, and an inbound user ID is not set, an inbound message from the foreign bus is only authorized to destinations that allow unauthenticated users access.
9. In the Foreign bus details pane, complete the details as appropriate:
   - If the service integration bus you want to connect to is in a different cell from the local bus, complete the following details:
     a. Ensure that **Configure a foreign bus in a remote cell** is selected.
     b. Enter the name of the service integration bus to connect to, that is, the foreign bus. Enter the exact name of the existing service integration bus.
     c. Enter the name of the gateway messaging engine in the foreign bus, that is, the messaging engine to connect to in the foreign bus.
     d. Select the **Configure publish-subscribe messaging for this connection** check box.
     e. Enter the name of the service integration bus link.
     f. Enter one or more bootstrap endpoints, that is, the host, port location, and transport chain for the messaging engine on the foreign bus that the local service integration bus connects to. The port is the SIB_ENDPOINT_ADDRESS (or SIB_ENDPOINT_SECURE_ADDRESS if security is enabled) of the messaging engine. Use the format *hostName*:*portNumber*:*chainName*, separating each bootstrap endpoint by a comma. For more information, see steps 1 and 2 of "Configuring a connection to a non-default bootstrap server" on page 1598. Although this task primarily describes how to configure a JMS connection factory, it is also applicable to setting several bootstrap endpoint values if the remote messaging engine is in a cluster.
   - If the service integration bus you want to connect to is in the same cell as the local bus, complete the following details:
     a. Ensure that **Configure a foreign bus in a local cell** is selected.
     b. Select the name of the service integration bus to connect to, that is, the foreign bus.
     c. Select the name of the gateway messaging engine in the foreign bus, that is, the messaging engine to connect to in the foreign bus.
     d. Select the **Configure publish-subscribe messaging for this connection** check box.
     e. Enter the name of the service integration bus link.
10. Optionally, to secure the connection, in the Foreign bus details pane, complete the following details:
    a. Select the **Secure connection** check box.
    b. Select the type of transport chain to use to communicate with the messaging engine in the foreign bus. Select one of the following:
       - InboundBasicMessaging. InboundBasicMessaging is a predefined transport chain where communication uses the TCP protocol.
       - InboundSecureMessaging. InboundSecureMessaging is a predefined transport chain where communication is secured by using the secure sockets layer (SSL) based encryption protocol over a TCP network. For successful connection, a set of suitably compatible SSL credentials must be associated with the local bus inbound channel and the foreign bus outbound channel.

- Other, please specify. Select this option to specify your own transport chain and enter the details in the field that appears.

c. Select the name of the authentication alias to use to authenticate access to the foreign bus. The alias must be known to the foreign bus.

11. In the Publish-subscribe details pane, repeat the following steps for each topic mapping you want to create:
    a. Select the name of the topic space on the local bus that will map to the topic space on the foreign bus.
    b. Enter the name of the topic space on the foreign bus. If the foreign bus is in the same cell as the local bus, you can select this name from a drop-down list.
    c. Click **Add**.
12. When the Foreign bus connection wizard is finished, save your changes to the master configuration.

Results

You have created a connection from a local service integration bus to a foreign service integration bus to use publish-subscribe messaging. You have created a direct foreign bus connection, which contains a routing definition, or virtual link. The physical link, a service integration bus link on the messaging engine for the local bus, is created automatically.

What to do next

You need to create a connection in the opposite direction between the two buses. To do this, repeat the procedure, using the bus you have just connected to as the local bus, and the bus you have just connected from as the foreign bus. Ensure that you use exactly the same name for the service integration bus link. After you create a foreign bus connection for each service integration bus, you can test the connection.

Connecting buses using an indirect connection:

You can connect a service integration bus to another service integration bus or a WebSphere MQ queue manager or queue-sharing group (known as the “gateway queue manager”) through one or more intermediate foreign buses. To do this, you create an indirect foreign bus connection.

Before you begin

To create an indirect foreign bus connection, the following resources must be defined:

- A service integration bus that you want to connect from, known as the local bus.
- A direct foreign bus connection from the local bus to the bus you want to use as the intermediate bus.
- A service integration bus or a gateway queue manager that you want to connect to indirectly, known as the target foreign bus.

About this task

To connect two buses indirectly, you create a indirect foreign bus connection on the local bus that specifies a suitable intermediate bus and the required target bus. There must be a connection, either direct or indirect, between the intermediate bus and the target foreign bus, that is, the service integration bus or a gateway queue manager that you want to connect to.

To connect one bus to another bus through an intermediate bus or a chain of buses, if the connection between the intermediate bus or the chain of buses and the target bus already exists, you do not require any new physical links. Instead, each foreign bus connection identifies a neighboring bus on the route to the final target bus as the ″next hop″ in the chain. Each bus in the chain must know about the next hop in the chain to reach the target bus. The local bus uses a foreign bus connection to identify the next bus in

the chain to the target bus, and uses its direct physical link to flow messages to that bus. Then, each intermediate bus uses its locally defined foreign bus connection to identify the next bus in the chain until the target bus is reached.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that you want to connect from, that is, the local bus.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**.
4. In the Foreign bus connections pane, click **New** to start the Foreign bus connection wizard.
5. In the Bus connection type pane, select **Indirect, using another bus**.
6. In the Indirect connection details pane, complete the following details:
   a. Enter the name of the target foreign bus that you want to connect to indirectly.
   b. Select the name of the existing, directly connected foreign bus that will be the intermediate bus to the target bus.
7. When the Foreign bus connection wizard is finished, save your changes to the master configuration.

Results

You have created a connection from the local bus to the target bus via the intermediate bus. You have created an indirect foreign bus connection, which contains a routing definition, or virtual link.

What to do next

Check whether there are connections between the intermediate bus you specified and the target bus. If necessary, repeat the procedure to create the ″next hop″ in the chain.

Testing foreign bus connections:

After you define a direct foreign bus connection between a service integration bus and either another service integration bus or a WebSphere MQ queue manager, you can test the connection to check that it can be used to exchange messages.

Before you begin

To test a foreign bus connection, the following resource must be defined:

- A service integration bus, known as the local bus, with a direct foreign bus connection to a service integration bus or a WebSphere MQ queue manager.

About this task

You might want to test a foreign bus connection before you send messages that use that connection, or if you have difficulty sending and receiving messages that use that connection. You can test only direct foreign bus connections. You cannot test indirect foreign bus connections, because it might not be possible to reach the target destination bus.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the Buses pane, click the bus that has the connection you want to test.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**. A list of foreign bus connections is displayed.
4. Select the check box for the connection you want to test.
5. Click **Test Connection**. A message is displayed at the top of the pane that shows the result of the test.

Results

You have tested a bus connection.

Listing the foreign bus connections:

You can list the foreign bus connections that are associated with a local bus to see which buses are currently configured.

Before you begin

To list foreign bus connections, the following resources must be defined:

- A service integration bus with one or more foreign bus connections between it and another service integration bus or a WebSphere MQ queue manager.

About this task

You can list foreign bus connections by using the administrative console, as described in this topic, or by using the wsadmin tool and the listSIBForeignBuses command.

To list the foreign bus connections that are associated with the local bus, complete the following steps.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses**.
3. In the service integration buses pane, select the bus for which you want to list the connections.
4. In the configuration tab, under **Topology**, click **Foreign bus connections**. A list of buses that have a foreign bus connection from the local bus is displayed.

Results

You have listed the foreign bus connections that are associated with the local bus.

Removing a foreign bus connection from a bus:

You can remove a foreign bus connection from a bus so that the connection can no longer be used for sending and receiving messages.

Before you begin

When a foreign bus connection is deleted from the configuration, the next time the hosting messaging engine for a link transmitter is started, it moves the messages to the exception destination. To avoid this situation, drain the messages from the link transmitter as far as is possible and then, before deleting the link configuration, either move any remaining messages from the transmission queues to an exception destination or delete them. For more details refer to the sub-topics below.

About this task

Removing a foreign bus connection from a bus removes the resources associated with the connection. The bus will no longer service the deleted link.

To remove a foreign bus connection:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the Buses pane, select the bus from which you want to remove the foreign connection.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**. to display a list of connections for this bus.

4. Select the check box that corresponds to the connection that you want to remove.
5. Click **Delete**.
6. Save your changes to the master configuration.

Results

You have now deleted the foreign bus connection from a bus. This connection can no longer be used for communication.

Preparing to remove a foreign bus connection between two service integration buses:

Perform this task before you remove a foreign bus connection.

Before you begin

Before you start you must know which foreign bus connection is being prepared for deletion.

About this task

When a foreign bus connection and its service integration bus link is deleted from the configuration, the next time the hosting messaging engine for a link transmitter is started, it deletes all its messages or moves them to the exception destination. To avoid messages being unintentionally deleted or moved to the exception destination, drain the messages from the link transmitter as much as possible and then, before deleting the link configuration, either move any remaining messages to an exception destination or delete them.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** to display a list of local buses.
3. Select the bus from which you want to remove the foreign bus connection.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**. A list of connections for this bus is displayed.
5. From the list of foreign buses, click the name of the foreign bus to display its details.
6. Under **General properties**, clear the **Send Allowed** check box to prevent new messages from being produced for this service integration bus link.
7. Click **Apply** to save the configuration.
8. To determine when there are no more messages queued on this link, under **Related Items** click **Service integration bus links** to display the details of the service integration bus links.
9. Click the **Refresh** icon of **Status** to refresh the view of the **Current outbound messages**.
10. When there are no **Current outbound messages**, select the check box next to the appropriate link and then click **Stop** to stop the link to the foreign bus.
11. When the **Status** of the link turns to red, the link to the foreign bus has no remaining messages and is stopped.
12. Repeat steps 2 to 11 on the foreign bus side because the foreign bus can continue to produce messages after the local bus connection has been deleted.
13. Save your changes to the master configuration.

Results

You have drained the messages from the link transmitter as much as possible, and either moved any remaining messages from the transmission queues to an exception destination or deleted them. You are now ready to remove the foreign bus connection.

Preparing to remove a foreign bus connection between a service integration bus and a WebSphere MQ network:

Perform this task before you remove a foreign bus connection.

Before you begin

Before you start you must know which foreign bus connection is being prepared for deletion. You should also inform the WebSphere MQ administrator that the foreign bus link is about to be deleted and therefore no longer paired with its WebSphere MQ gateway queue manager or message broker in the WebSphere MQ network.

About this task

When a foreign bus link is deleted from the configuration, the next time the hosting messaging engine for a link transmitter is started it deletes all its messages or moves them to the exception destination. To avoid messages being unintentionally deleted or moved to the exception destination, drain the messages from the link transmitter as much as possible and then, before deleting the link configuration, either move any remaining messages to an exception destination or delete them.

If there are publish/subscribe broker profiles defined, you should remove the subscriptions.

1. Start the administrative console.
2. Optional: If there are publish/subscribe broker profiles defined on any of the links for this foreign bus connection, remove the subscriptions.

   Complete the following substeps for each broker profile:

   a. Navigate to **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles** → ***profile_name***
   b. Click the **Runtime** tab.
   c. Click **Subscriptions**.
   d. Click **Unsubscribe** to remove all the subscriptions listed.

   When an unsubscribe command is sent to the message broker in the WebSphere MQ network, the relevant topic mapping is put into an ″in doubt″ state until the unsubscribe is confirmed when the topic mapping is deleted. After the unsubscribe is confirmed the topic mapping is no longer shown in the runtime view. You might need to refresh the runtime view for all subscriptions to be shown as removed.
3. Prevent new messages from being produced for this foreign bus connection.
   a. Navigate to **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name***
   b. Under **General properties**, clear the **Send Allowed** check box.
   c. Click **Apply** to save the configuration.
4. Determine when there are no more messages queued, then stop the link to the foreign bus in a controlled manner.
   a. Under **Related Items**, click **WebSphere MQ links** to display the list of links for this bus.
   b. Click the **Refresh** icon of **Status** to refresh the view of the **Current outbound messages**.
   c. When there are no **Current outbound messages**, select the check box next to the appropriate link and then select a **Stop mode** of **Quiesce**.
   d. Select a **Target state** of **Stopped** so that the link can only be started again by administrator action.
   e. When the **Status** of the link turns to red, the link to the foreign bus has no remaining messages and is stopped.

5. The foreign bus can continue to produce messages after the local bus connection has been deleted. Because the foreign bus is a WebSphere MQ network, refer to the WebSphere MQ Intercommunication guide for details about the safe deletion of channels at Managing WebSphere MQ channels.
6. Save your changes to the master configuration.

Results

You have removed the subscriptions from any publish/subscribe brokers on the link. You have drained the messages from the link transmitter as much as possible, and either moved any remaining messages from the transmission queues to an exception destination or deleted them.

What to do next

You are now ready to remove the foreign bus connection.

Configuring destination defaults for a foreign bus connection:

You can use destination defaults to configure the destination for messages handled by a foreign bus connection. A foreign bus connection can be either a service integration bus or a WebSphere MQ queue manager or queue-sharing group (known as a “gateway queue manager”).

Before you begin

To configure destination defaults for a foreign bus connection, the following resource must be defined:

- At least one foreign bus connection, configured

About this task

Destination defaults are used where an application is producing messages destined for a destination on a foreign bus connection, but where that application has not explicitly set certain attributes on the message, or where a foreign destination has not been explicitly defined.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the service integration buses pane, select the bus that you want to configure.
3. In the configuration tab, under **Topology**, click **Foreign bus connections**. A list of foreign bus connections is displayed.
4. Select the foreign bus connection that you want to configure.
5. In the content pane, under **Additional properties**, click **Destination defaults**.
6. Specify the following properties for the destination defaults:

   Default priority
   The default priority assigned to messages sent to this destination when a priority has not been set by the producer.

   Default reliability
   The reliability assigned to a message produced to this destination when an explicit reliability has not been set by the producer.

   Maximum reliability
   The maximum reliability of messages accepted by this destination.

   Send allowed
   Clear this option (setting it to false) to stop producers from being able to send messages to destinations on this foreign bus.

Enable producers to override default reliability
Select this option to enable producers to override the default reliability that is set on the destination.

7. Click **OK**.
8. Save your changes to the master configuration.

Results

You have now configured destination defaults for a foreign bus connection.

Managing messages on foreign buses:

This task describes how to manage messages on a foreign bus.

Before you begin

The steps required depend on whether the foreign bus connection connects a service integration bus to:

- another service integration bus
- a WebSphere MQ network

About this task

If the foreign bus connection connects a service integration bus to another service integration bus, you can manage messages on:

- link transmitters
- link receivers

If the foreign bus connection connects a service integration bus to a WebSphere MQ network, you can manage messages on:

- link transmitters
- known link transmitters
- sender channel transmitter

There is one link transmitter for each message engine, one known link for each link transmitter, and one sender channel transmitter on the WebSphere MQ link sending messages to a gateway queue manager on the WebSphere MQ network acting as the foreign bus.

Perform this task when you want to view, delete, or move messages on a transmitter.

Refer to the list of sub-topics below and select the appropriate task.

Results

You have managed the messages on a foreign bus.

Configuring an exception destination for messages on a foreign bus connection:

You can configure an exception destination for messages handled by a foreign bus connection. A foreign bus connection can be either a connection between two service integration buses or a connection between a service integration bus and a WebSphere MQ network.

Before you begin

To configure an exception destination default for a foreign bus connection, you must know the name of the foreign bus connection.

About this task

If specified, the link exception destination is used as the target for messages being exceptioned for that link. The exception destination can be specific to that link, the system default local messaging engine exception destination or, if the exception destination is specified as **None**, the messages are discarded.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** to display a list of local buses.
3. Select the bus from which you want to remove the foreign bus connection.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections** to display a list of connections for this bus.
5. From the list of foreign buses, click the name of the foreign bus to display its details.
6. Under **Related Items** click **Service integration bus links** (if the foreign bus is another service integration bus) or **WebSphere MQ links** (if the foreign bus is WebSphere MQ network).
7. From the list of links, click the name of a link to display its details.
8. In the **Configuration** tab, under **General properties**, use the radio buttons to configure the **Exception destination** to which error messages are sent for this link. Select **None** to discard error messages. Select **System** to send error messages to the system default local messaging engine exception destination. Select **Specify** and enter an exception destination to send error messages to the specified exception destination.
9. Save your changes to the master configuration.

Results

You have now configured an exception destination for messages on a foreign bus connection.

Managing messages in a link transmission queue connection between two service integration buses:

This task describes how to manage messages in a link transmission queue connected to a foreign bus connection between two service integration buses.

Before you begin

You must know the link transmitter from which messages are to be removed.

About this task

There is one link transmitter for each message engine.

Perform this task when you want to view, delete or move messages on a link transmitter.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
3. Select the bus whose link transmission queue you want to manage.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**. A list of connections for this bus is displayed.
5. Select the **Name** of the foreign bus from the list of foreign buses. The details of the selected foreign bus are displayed in the content pane.

6. In the **Configuration** tab, under **Related Items**, click **Service integration bus links**, then select the name of the link.
7. In the **Runtime** tab, click **Link transmitters** to display the list of link transmitters for messaging engines that are sending messages to the foreign bus.
8. Optional: To manage all messages on the link transmitter, select the **Link transmitter** and perform your required task:
   - Click **Delete all messages** to delete all the available messages on the selected link transmitter.
   - Click **Move all messages** to move all the available messages on the selected link transmitter to the configured exception destination. If the exception destination is configured as **None** the messages are discarded. If it is configured as **System default** the messages are sent to the system default exception destination. If a **Specified destination** is configured, the messages are sent to the exception destination specified for this link.
9. Optional: To manage selected messages on the link transmitter, click the name of the **Messaging engine** from the list of **Link transmitters** and then click **View messages** to view the messages on an outbound stream from this link transmitter to the foreign bus.
10. Select one or more messages and :
    - Click **Delete** to delete the selected messages.
    - Click **Move** to move the selected messages to the local exception destination.
    - Click **Refresh** to view the current messages on the selected link transmitter.

      Note: Only messages up to the specified **Maximum displayed messages** are displayed.

Results

You have managed the messages on a specified link transmitter connected to a foreign bus.

Managing messages in a link receiver queue connected to a foreign bus:

This topic describes how to manage messages in a link receiver queue connection between two service integration buses.

Before you begin

There is one link receiver for each message producing foreign messaging engine on the service integration bus link.

About this task

Perform this task when you want to view messages on a link receiver.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** to display a list of buses.
3. Click the bus whose link receiver queue you want to view.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**.
5. Click the name of the foreign bus connection to display its details.
6. Under **Related links** click **Service integration bus links** and click the name of the link.
7. In the **Runtime** tab, click **Link receivers** to display the list of link receivers for messaging engines that are sending messages to this bus.
8. To view the messages on the link receiver, click the name of the **Messaging engine** from the list of link receivers and then click **View messages** to view the messages on an inbound stream from the messaging engine on the foreign bus. Click **Refresh** to view the current messages on the selected link receiver.

Results

You have viewed the messages on a specified link receiver connected to a foreign bus.

Managing messages in a link transmission queue connection between a bus and a WebSphere MQ network:

This task describes how to manage messages in a link transmission queue connected to a foreign bus connection between a service integration bus and a WebSphere MQ network.

Before you begin

You must know the foreign bus whose messages are to be removed.

About this task

There is one link transmitter for each message engine producing messages on the foreign bus link.

Perform this task when you want to view, delete or move messages on a link transmitter.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed in the content pane.
3. Select the bus whose link transmission queue you want to manage.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**. A list of connections for this bus is displayed.
5. Select the **Name** of the foreign bus from the list of foreign buses. The details of the selected foreign bus are displayed in the content pane.
6. In the **Configuration** tab, under **Related Items**, click **WebSphere MQ links**, then select the **Name** of the link.
7. In the **Runtime** tab, click **Sender channel** to display the sender channels that are sending messages to the foreign bus.
8. Optional: To manage all messages on the link transmitter, select the **Link transmitter** and perform your required task:
   - Click **Delete all messages** to delete all the available messages on the selected link transmitter.
   - Click **Move all messages** to move all the available messages on the selected link transmitter to the configured exception destination. If the exception destination is configured as **None** the messages are discarded. If it is configured as **System default** the messages are sent to the system default exception destination. If a **Specified destination** is configured, the messages are sent to the exception destination specified for this link.
9. Optional: To manage selected messages on the link transmitter, click the name of the **Messaging engine** from the list of **Link transmitters** and then click **View messages** to view the messages on an outbound stream from this link transmitter to the foreign bus.
10. Select one or more messages and :
    - Click **Delete** to delete the selected messages.
    - Click **Move** to move the selected messages to the local exception destination.
    - Click **Refresh** to view the current messages on the selected link transmitter.

      Note: Only messages up to the specified **Maximum displayed messages** are displayed.

Results

You have managed the messages on a link transmitter connected to a foreign bus.

Managing messages in a known link transmission queue connected to a foreign bus:

How to manage messages in a known link transmission queue connection between a service integration bus and a WebSphere MQ network.

Before you begin

There is one known link transmitter for each message producing messaging engine on the foreign bus link.

About this task

Perform this task when you want to manage messages on a known link transmitter.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** to display a list of buses.
3. Click the bus whose known link transmission queue you want to view.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**. to display a list of connections for this bus.
5. From the list of foreign buses, click the name of the foreign bus to display its details.
6. Under **Related Items** click **WebSphere MQ links** and click the **Name** of the link.
7. In the **Configuration** tab, under **Additional Properties** click **Sender channel** to display the sender channels for all messaging engines that are sending messages to this bus. The sender channel sends messages to the gateway queue manager.
8. From the list of sender channels, select the check box of a **Sender MQ channel name** and click **View known link transmitters** to display a list the remote messaging engines producing messages for this WebSphere MQ link.
9. Click **Refresh** to view the current messages on the message engines.
10. Select the check box of the known link transmitter and:
    - Click **Delete all messages** to remove them from the known link transmission queue.
    - Click **Move all messages** to move all the available messages on the known link transmitter to the configured exception destination. If the exception destination is configured as **None** the messages are discarded. If it is configured as **System default** the messages are sent to the system default exception destination. If a **Specified destination** is configured, the messages are sent to the exception destination specified for this link.

Results

You have managed the messages on known link transmitters producing messages to a sender channel transmitter connected to a foreign bus.

Managing messages in a sender channel transmission queue connected to a foreign bus:

How to manage messages in a sender channel transmission queue connection between a service integration bus and a WebSphere MQ network.

Before you begin

There is one sender channel transmitter on the foreign bus link sending messages to a gateway queue manager on the WebSphere MQ network acting as the foreign bus.

About this task

Perform this task when you want to manage messages on the sender channel transmitter.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** to display a list of buses.
3. Click the bus whose sender channel transmission queue you want to view.
4. In the **Configuration** tab, under **Topology**, click **Foreign bus connections**. to display a list of connections for this bus.
5. From the list of foreign buses, click the name of the foreign bus to display its details.
6. In the **Configuration** tab, under **Related Items** click **WebSphere MQ links** and select the check box for a link.
7. Click **Sender channel transmitter** under **Current outbound messages** or click **Sender channel transmitter** under **Messages sent** to see the relevant message details for this sender channel transmitter.
8. Select the check box of the sender channel transmitter and:
   - Click **Delete all messages** to remove them from the sender channel transmission queue.
   - Click **Move all messages** to move all the available messages on the sender channel transmitter to the configured exception destination. If the exception destination is configured as **None** the messages are discarded. If it is configured as **System default** the messages are sent to the system default exception destination. If a **Specified destination** is configured, the messages are sent to the exception destination specified for this link.

Results

You have managed the messages on a sender channel transmitter sending messages to a gateway queue manager on the WebSphere MQ network acting as the foreign bus.

Managing pending acknowledgement messages on a deleted WebSphere MQ link:

This topic describes how to manage pending acknowledgement messages on a WebSphere MQ link that has been deleted from a foreign bus connecting to a service integration bus.

Before you begin

Before you start you need to know the name of the WebSphere MQ link that has been deleted.

About this task

If a foreign bus link to a WebSphere MQ network is deleted from the WebSphere Application Server configuration before being drained of messages, a batch of messages pending acknowledgement remains persisted in the WebSphere MQ link sender channel transmitter.

To resolve pending acknowledgement messages on a deleted WebSphere MQ link, use the administrative console to complete the following steps.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses**. to display a list of buses.
3. Click the bus whose link transmission queue you want to manage.
4. In the **Configuration** tab, under **Topology**, click **Foreign Bus Connections** to display a list of connections for this bus.
5. From the list of foreign buses, click the name of the foreign bus to display its details.
6. Under **Related Items** click **Service integration bus links** to display the details of the service integration bus links.

7. Click the MQ network foreign bus that has a connection that is active, but a configuration status that is **Deleted**. If clicking **Link transmitters** displays an empty list, no messaging engines are producing messages to this link and all the link transmitters have been deleted because they were drained of messages. The **Sender channel transmitter** link displays the **Status** of the sender channel as stopped but **Current outbound messages** shows remaining messages on the sender channel transmitter.
8. Click a WebSphere MQ link **sender channel** link to display the messages that are queued on the WebSphere MQ link sender channel transmitter for transmission to the WebSphere MQ network.
9. If the **Status** of a batch of messages is **Commit pending batch**, the batch has arrived safely at the MQ network. Select the batch and click **Commit pending batch** to remove the messages from the transmission queue.
10. If the **Status** of a batch of messages is **Pending batch acknowledgement**, the batch did not arrive at the MQ network. Select the batch and click **Rollback pending acknowledge batch** to rollback the transaction and restore the messages to the channel in an available state. These messages are either automatically deleted or moved to the exception destination. When the channel transmitter is empty, the link is automatically deleted from the runtime environment.

Results

You have resolved any pending acknowledgement messages on a WebSphere MQ link that has been deleted from a foreign bus connecting to a service integration bus.

Modifying a routing definition:

You can view or change the routing definition of an existing foreign bus connection between a local bus and a foreign bus. The routing definition defines the virtual link between two buses that enables them to exchange messages. The routing definition can define properties for a virtual service integration bus link, a virtual WebSphere MQ link, or an indirect foreign bus connection.

About this task

You create a routing definition when you create a foreign bus connection. For a direct foreign bus connection, the routing definition, or virtual link, has a corresponding physical link on a messaging engine.

To change the properties of the routing definition, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that is connected to another bus, that is, the local bus you require.
3. In the content pane, under **Topology**, click **Foreign bus connections**. A list of buses that have a foreign bus connection from the local bus is displayed.
4. Select the required foreign bus.
5. In the content pane, under **Additional properties**, click the routing properties option. The routing properties option depends on the type of routing definition and is one of the following:
   - **Service integration bus link routing properties**
   - **WebSphere MQ link routing properties**
   - **Indirect routing properties**
6. Specify the new routing properties:
   - For a virtual service integration bus link or a virtual WebSphere MQ link, specify properties as follows:

Inbound user ID

The user name used to authenticate inbound message flows from the foreign bus.

When the local bus is secure, the inbound user ID replaces the user ID in messages from the foreign bus that arrive at the local bus and is used to authorize whether those messages can access their destinations. Specify an inbound user ID for the local service integration bus under the following circumstances:

– The foreign bus is in a different security domain, so user IDs in the foreign bus are not recognized in the local bus.

– You want local control of access to inbound messages to the local bus.

If the local bus is not secure, the inbound user ID has no effect on messages. If the local bus is secure, the foreign bus is not secure, and an inbound user ID is not set, an inbound message from the foreign bus is only authorized to destinations that allow unauthenticated users access.

Outbound user ID

The user name used to authenticate outbound message flows to the foreign bus.

The outbound user ID replaces the user ID that identifies the message source in every message sent to the foreign bus. Where it is defined, the outbound user ID replaces the user ID in messages sent by the local bus to the foreign bus. If the local bus and the foreign bus are both secure, and the foreign bus has not overridden the user ID with its own inbound user ID, the foreign bus also uses the outbound user ID to authorize the message to its destination.

- For an indirect foreign bus connection, select the name of the next foreign bus you require in the chain of intermediate buses. For a bus to be available for selection, it must already have a direct foreign bus connection from the local bus.

7. Click **OK**.
8. Save your changes to the master configuration.

Configuring service integration bus links:

You can configure service integration bus links on messaging engines in a variety of ways. For example, you can start, stop, or remove links.

About this task

When you create a foreign bus connection to connect two service integration buses, a service integration bus link is created automatically. The foreign bus connection contains a routing definition, which is a virtual link, and the service integration bus link is the corresponding physical link on the messaging engine.

Configuring the properties of a service integration bus link:

After establishing a service integration bus link you might want to configure the properties of a service integration bus link such as the name of the service integration bus link, or authentication alias used by foreign bus that the link connects to.

About this task

To configure the properties of a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to configure the service integration bus link.

3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you want to configure.
5. Specify the following properties for the service integration bus link:

   Name The name of the service integration bus link. In order to work, the name must be the same as the name of the corresponding service integration bus link configured on the target foreign bus.

   Description
   An optional description for the service integration bus link, for administrative purposes.

   UUID The universal unique identifier assigned by the system to the service integration bus link for administrative purposes.

   Foreign messaging engine
   The messaging engine on the foreign bus to which this link connects.

   Note: This foreign bus name must not be altered after it has been configured. If you alter it, any messaging engines that already hold state information about the link will not be able to use the link unless the foreign bus name is reset to its original value.

   Target inbound transport chain
   The type of transport chain used for communication with the foreign bus. The transport chain name must be the name of the transport chain as defined on the server on which the target messaging engine is hosted.

   Bootstrap endpoints
   The comma-separated list of endpoints used to connect to a bootstrap server. This property is set in the same way as the **Provider endpoint** property in the JMS connection factory settings. For more information, see steps 1 and 2 of “Configuring a connection to a non-default bootstrap server” on page 1598. Although this task primarily describes how to configure a JMS connection factory, it is also applicable to setting several bootstrap endpoint values if the remote messaging engine is in a cluster.

   Note: Service integration bus links over BootstrapTunneledMessaging and BootstrapTunneledSecureMessaging transport chains only work directly between application server instances. Bus links over TunneledMessaging transport chains do not work if an HTTP server is placed in front of either application server instance.

   Authentication alias
   The name of the authentication alias, used to authenticate access to the foreign bus. The alias must be known to the foreign bus.

   Initial state
   Whether the link is started automatically when the messaging engine is started. Until started, the gateway link is unavailable. If this property is set to **Started** the service integration bus link is started when the messaging engine is started.
6. Click **OK**.
7. Save your changes to the master configuration.

Listing the service integration bus links:

You can list all the service integration bus links that are linked to a messaging engine.

About this task

To list the service integration bus links for a messaging engine, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.

2. In the content pane, click the name of the bus that your messaging engine belongs to.
3. In the content pane, under **Additional properties**, click **Messaging engines**. The list of messaging engines in the bus is displayed.
4. In the content pane, select the messaging engine for which you want to list the service integration bus links.
5. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.

Example

The following combinations of **Status** and **Activity** values are possible:

Status	Activity	Meaning
started	inactive	The service integration bus link is started on the local messaging engine but has no connection to the foreign bus. The service integration bus link is attempting to activate a connection to the foreign bus. The service integration bus link on the foreign bus must also be started to successfully activate of a connection between the buses.
started	active	The service integration bus link is started on the local messaging engine and has an active connection to the foreign bus.
stopped	inactive	The service integration bus link is stopped on the local messaging engine and there is no connection to the foreign bus.
unknown	inactive	An error might have occurred in setting up the link, such that the object that is used to report the current state is not available.

What to do next

You can now add or remove a service integration bus link or select a service integration bus link to start, stop, or configure.

Starting a service integration bus link:

When a service integration bus link has been started, it can be used for communicating with its associated messaging engines.

About this task

To start a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to start the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to start.
5. Click **Start**.

Stopping a service integration bus link:

When a service integration bus link has been stopped, it cannot be used for communication until it is restarted.

About this task

To stop a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine to which the service integration bus link that you wish to stop belongs.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to stop.
5. Click **Stop**.

Removing a service integration bus link:

When a service integration bus link to a messaging engine has been removed, it cannot be used for communication until it is recreated.

Before you begin

Before you remove the service integration bus link you must stop the link.

When you remove a service integration bus link, all traffic using the link needs to be dealt with in a similar way to when you remove a destination. For further details, see “Deleting a non-topic space bus destination” on page 1246. Note that express Quality of Service (QoS) messages are discarded.

About this task

To remove a service integration bus link from a messaging engine, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select which the messaging engine has own association with the service integration bus link that you wish to remove.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to remove.
5. Click **Delete**.
6. Save your changes to the master configuration.

What to do next

When you remove a service integration bus link, all traffic using the link must be diverted by alternative routes or held pending further administrative action. Express QoS messages are discarded.

Configuring topic space mappings between service integration buses:

A topic space mapping allows subscribers on the local topic space to receive messages published in the foreign topic space. You can configure topic space mappings between a local service integration bus and a foreign service integration bus.

About this task

A topic space mapping allows subscribers on the local topic space to receive messages published in the foreign topic space. For publications to flow from the local topic space into the foreign bus, an equivalent topic space mapping is required by the foreign bus.

If you want to publish messages from a local bus to a foreign bus that is linked indirectly through a third bus, you must configure topic space mappings between the local bus and the intermediate bus, and between the intermediate bus and the foreign bus. The messages can be for publication to the foreign bus only, or to all three buses. If you want to publish messages only to the local and foreign buses, these two buses must be connected directly.

If you want to publish messages from a local service integration bus to a foreign bus that represents a WebSphere MQ network, you must define topic mappings on the broker profile associated with the WebSphere MQ link.

For more details about the foreign bus and publish/subscribe messaging between buses, see Foreign buses.

Creating topic space mappings:

A topic space mapping allows subscribers on the local topic space to receive messages published in the foreign topic space. You can create topic space mappings between two service integration buses.

About this task

You can create topic space mappings as part of the procedure to create a foreign bus connection, or you can create them separately, as described in this topic.

Note that for publishing and subscribing between a service integration bus and a foreign bus that represents a WebSphere MQ network, you must define topic mappings on the broker profile associated with the WebSphere MQ link.

To map a topic space on a local service integration bus to a topic space on a foreign service integration bus, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that is connected to another bus, that is, the local bus you require.
3. In the content pane, under **Topology**, click **Foreign bus connections**. A list of buses that have a foreign bus connection from the local bus is displayed.
4. Select the required foreign bus.
5. In the content pane, under **Additional properties**, click **Service integration bus link routing properties**.
6. In the content pane, under **Additional properties**, click **Topic space map entries**.
7. Click **New**.
8. Specify the mappings as follows:

   Local topic space
   The name of the topic space on this (local) bus that is mapped to the remote topic space on the foreign bus.

   Remote topic space
   The name of the topic space on the foreign bus that is mapped to the local topic space.

   Note: The names of the local and foreign topic space do not have to match, but the names of the topics must match in both local and foreign buses.
9. Click **OK**.
10. Save your changes to the master configuration.

Results

The remote topic space is mapped to the local topic space.

Deleting topic space mappings:

You can delete topic space mappings so that subscribers on the local topic space cannot receive messages that are published in the foreign topic space.

About this task

To delete topic space mappings, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that is connected to another bus, that is, the local bus you require.
3. In the content pane, under **Topology**, click **Foreign bus connections**. A list of buses that have a foreign bus connection from the local bus is displayed.
4. Select the required foreign bus.
5. In the content pane, under **Additional properties**, click **Service integration bus link routing properties**.
6. In the content pane, under **Additional properties**, click **Topic space map entries**. A list of topic space map entries is displayed.
7. Select the topic space map entry that you wish to remove.
8. Click **Delete**.
9. Save your changes to the master configuration.

Listing topic space map entries:

A topic space mapping allows subscribers on the local topic space to receive messages published in the foreign topic space. You can list topic space map entries to view existing map entries.

Before you begin

There must be a service integration bus link between a local service integration bus and a foreign service integration bus.

About this task

To list the topic space map entries, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**. A list of service integration buses is displayed.
2. In the Buses pane, click the service integration bus that is connected to another bus, that is, the local bus you require.
3. In the content pane, under **Topology**, click **Foreign bus connections**. A list of buses that have a foreign bus connection from the local bus is displayed.
4. Select the required foreign bus.
5. In the content pane, under **Additional properties**, click **Service integration bus link routing properties**.
6. In the content pane, under **Additional properties**, click **Topic space map entries**. The list of mappings between topic spaces in the local bus and topic spaces in the foreign bus is shown.

Topic names and use of wildcard characters in topic expressions:

This topic describes the use of wildcard characters in topic expressions to retrieve topics provided by the default messaging provider and service integration technologies.

Each subscribe request includes a topic expression that identifies one or more topics that the subscription is to be associated with, and which the request uses to match against incoming messages.

Subscription topic expressions for the default messaging provider and service integration technologies are based on a subset of the XPath location path syntax.

Identifying individual topics

Every topic in a topic space has a *topic name* consisting of one or more name parts, separated by / (forward slash) characters:

```
Topic name = name_part | (name_part '/' topic_name)
```

Using wildcards to identify multiple topics

To select one or more topics in a topic space, you can use a *topic path*, a location path that contains wildcard characters. Topic spaces are evaluated using a subset of the XPath location path syntax with the <topicspace> element as the initial context node, so that non-wildcarded topic paths look exactly like topic names.

The syntax for topic paths can be summarized as follows:

- A topic path that contains no * (asterisk), // (double forward slash), or . (dot) symbols is asking for an exact match with the topic name specified.
- A * (asterisk) can be used as a wild card and matches one level (regardless of the value of the name part at that level)

  A * can be used anywhere in a topic path expression, but if it isn't at the start it must be preceded by a /, and if it is isn't at the end it must be followed by a /
- // can be used as a wild card and matches 0 or more levels

  A // can be used anywhere in the expression except at the end. To match 0 or more levels at the end of the expression you end the expression with the syntax //. (double-slash dot). To match one or more levels at the end use //* (double-slash asterisk)

  A topic path must not contain more than two consecutive / symbols.

The following table lists some example topic paths using the XPath syntax and the equivalent WBI Message Broker selectors:

Topic path	Topics selected	WBI Message Broker equivalent
A/B	Selects the B child of A	A/B
A/*	Selects all children of A	A/+
A//*	Selects all descendents of A	A/#/+
A//.	Selects A and all descendents of A	A/#
//*	Selects everything	# (or #/+)
A/./B	Equivalent to A/B	A/B
A/*/B	Selects all B grandchildren of A	A/+/B
A//B	Selects all B descendents of A	A/#/B
//A	Selects all A elements at any level	#/A

*	Selects all first level elements	+

Interoperation with Version 5.1 clients

Existing WebSphere Application Server Version 5.1 client applications using Version 5.1 connection factory and destination definitions use the WBI Message Broker wildcard convention. Such applications can connect to the default messaging provider and service integration bus, and automatically have their wildcard syntax mapped to the XPath convention when subscriptions are created. Any display of these subscriptions through an administrative interface to the bus shows the XPath syntax.

Defining outbound chains for bootstrapping

You can define new outbound chains using the wsadmin utility. These chains can be used for bootstrapping connections to messaging engines.

About this task

To create outbound chains for bootstrapping, there are involves several main steps:

1. Locate the appropriate TransportChannelService configuration object. This object is the parent object of all the objects created.
2. Create the individual channels that comprise the transport channel service. Some of these channels might require references to other configuration objects.
3. Assemble the channels that you have created into an outbound channel chain.

The channels used to build an outbound bootstrap chain determine the protocol with which the outbound chain can be used to bootstrap. The following table shows all valid bootstrap chains with their bootstrap protocols.

Bootstrap protocol	TCP channel	SSL channel	HTTP channel	HTTP tunneling channel	JFAP channel
TCP	X				X
SSL	X	X			X
HTTP	X		X	X	X
HTTPS	X	X	X	X	X

For example, a chain for bootstrapping using the SSL protocol would consist of a TCP channel, SSL channel, and JFAP channel. When you create chains, the order of channels in the chain is important. You must specify channels in the order (left to right) in which they appear in the preceding table.

The example in this topic describes how to create a bootstrap chain that is capable of bootstrapping using the HTTPS protocol. This requires a chain containing all the channel types described. Thus, it is easy to see how to create chains for other protocols by omitting channels during the chain creation step.

Note: You open a wsadmin command session from within Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

1. Locate the TransportChannelService object for the server in which you wish to create the new chain. For example, in a WebSphere Application Server Network Deployment configuration, you can list the available TransportChannelService objects and select the appropriate service as follows.

   Using Jython:

```
wsadmin>AdminConfig.list("TransportChannelService" )
(cells/BadgerCell01/nodes/BadgerCellManager01/servers/dmgr|server.xml
#TransportChannelService_1)
(cells/BadgerCell01/nodes/BadgerNode01/servers/nodeagent|server.xml
#TransportChannelService_1095
711814579)
```

```
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml
#TransportChannelService_109571
2023139)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server2|server.xml
#TransportChannelService_109571
2039302)
wsadmin>tcs = AdminConfig.list("TransportChannelService" ).split("\r\n")[2]
```

Using Jacl:

```
wsadmin> $AdminConfig list TransportChannelService
(cells/BadgerCell01/nodes/BadgerCellManager01/servers/dmgr|server.xml
#TransportChannelService_1)
(cells/BadgerCell01/nodes/BadgerNode01/servers/nodeagent|server.xml
#TransportChannelService_1095
711814579)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml
#TransportChannelService_109571
2023139)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server2|server.xml
#TransportChannelService_109571
2039302)
wsadmin> set tcs [lindex [$AdminConfig list TransportChannelService] 2]
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
TransportChannelService_1095712023139)
```

2. Define an outbound TCP channel called testTCPChannel.

   Using Jython:

```
wsadmin>tcpChannel = AdminConfig.create("TCPOutboundChannel", tcs,
[["name", "testTCPChannel"]] )
```

   Using Jacl:

```
wsadmin>set tcpChannel [$AdminConfig create TCPOutboundChannel $tcs
"{name testTCPChannel}"]
testTCPChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/
server1|server.xml#TCPOutboundChannel_1095969213949)
```

3. Define an outbound SSL channel called testSSLChannel. There are two steps required to define such a channel.

   a. Identify the SSL alias to be used by the channel.

      Using Jython:

```
wsadmin>for obj in AdminConfig.list("SSLConfig" ).split("\r\n"):
print obj+AdminConfig.show(obj, "alias")
(cells/BadgerCell01|security.xml#SSLConfig_1)
[alias BadgerCellManager01/DefaultSSLSettings]
(cells/BadgerCell01|security.xml#SSLConfig_1095711819776)
[alias BadgerNode01/DefaultSSLSettings]
```

      Using Jacl:

```
wsadmin>foreach obj [$AdminConfig list SSLConfig] { puts "$obj
[$AdminConfig show $obj alias]]" }
(cells/BadgerCell01|security.xml#SSLConfig_1) {alias BadgerCellManager01/
DefaultSSLSettings}]
(cells/BadgerCell01|security.xml#SSLConfig_1095711819776) {alias BadgerNode01/
DefaultSSLSettings}]
```

   b. Create an SSL channel as in the following example, in which the BadgerNode01/DefaultSSLSettings alias is used.

      Using Jython:

```
wsadmin>sslChannel =
AdminConfig.create("SSLOutboundChannel", tcs, [["name", "testSSLChannel"],
["sslConfigAlias", "BadgerNode01/DefaultSSLSettings"]] )
```

      Using Jacl:

```
wsadmin>set sslChannel [$AdminConfig create SSLOutboundChannel $tcs
"{name testSSLChannel}
{sslConfigAlias BadgerNode01/DefaultSSLSettings}"]
testSSLChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
SSLOutboundChannel_1095971760671)
```

4. Define an outbound HTTP channel called testHTTPChannel.

   Using Jython:

```
wsadmin>httpChannel = AdminConfig.create("HTTPOutboundChannel", tcs,
[["name", "testHTTPChannel"]] )
```

Using Jacl:

```
wsadmin>set httpChannel [$AdminConfig create HTTPOutboundChannel $tcs
"{name testHTTPChannel}"]
testHTTPChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
HTTPOutboundChannel_1095971896556)
```

5. Define an outbound HTTP tunneling channel called testHTCChannel.

   Using Jython:

```
wsadmin>htcChannel = AdminConfig.create("HTTPTunnelOutboundChannel", tcs,
[["name", "testHTCChannel"]] )
```

   Using Jacl:

```
wsadmin>set htcChannel [$AdminConfig create HTTPTunnelOutboundChannel $tcs
"{name testHTCChannel}"]
testHTCChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
HTTPTunnelOutboundChannel_1095972164201)
```

6. Define an outbound JFAP channel called testJFAPChannel.

   Using Jython:

```
wsadmin>jfapChannel = AdminConfig.create("JFAPOutboundChannel", tcs,
[["name", "testJFAPChannel"]] )
```

   Using Jacl:

```
wsadmin>set jfapChannel [$AdminConfig create JFAPOutboundChannel $tcs
"{name testJFAPChannel}"]
testJFAPChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
JFAPOutboundChannel_1095972226631)
```

7. Finally, create the channel chain by combining the channels defined so far. For example, to create a chain called testChain:

   Using Jython:

```
wsadmin>AdminConfig.create("Chain", tcs, [["name", "testChain"], ["enable", "true"],
["transportChannels", [tcpChannel, httpChannel, htcChannel, jfapChannel]]] )
testChain(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
Chain_1095972662147)
```

   Using Jacl:

```
wsadmin>$AdminConfig create Chain $tcs "{name testChain} {enable true}
{transportChannels {$tcpChannel $httpChannel $htcChannel $jfapChannel}}"
testChain(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
Chain_1095972662147)
```

Defining outbound chains for WebSphere MQ interoperation

You can define new outbound chains using the wsadmin utility. These chains can be used for interoperating with WebSphere MQ.

About this task

The channels used to build an outbound chain determine with which configurations of the WebSphere MQ queue manager sender channel a network connection can be successfully established. The following table shows all the valid chain configurations and describes the configuration of WebSphere MQ queue manager sender channel with which they can be used to establish a connection.

WebSphere MQ channel	TCP channel	SSL channel	MQFAP channel
Unsecured WebSphere MQ channel	X		X
WebSphere MQ channel secured using SSL	X	X	X

For example, an SSL-based chain would consist of a TCP channel, SSL channel and MQFAP channel. When creating chains, the order of channels in the chain is important. You must specify channels in the order (left to right) in which they appear in the above table.

The example in this topic describes how to create an outbound chain capable of being used to contact WebSphere MQ queue manager receiver channels using SSL-based encryption.

Note: You open a wsadmin command session from within Qshell. For more information, see the topic "Configure Qshell to run WebSphere Application Server scripts".

1. Locate the TransportChannelService object for the server in which you wish to create the new chain. For example, in a WebSphere Application Server Network Deployment configuration, you can list the available TransportChannelService objects and select the appropriate service.

   Using Jython:

```
wsadmin>AdminConfig.list("TransportChannelService" )
(cells/BadgerCell01/nodes/BadgerCellManager01/servers/dmgr|server.xml
#TransportChannelService_1)
(cells/BadgerCell01/nodes/BadgerNode01/servers/nodeagent|server.xml
#TransportChannelService_1095
711814579)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml
#TransportChannelService_109571
2023139)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server2|server.xml
#TransportChannelService_109571
2039302)
wsadmin>tcs = AdminConfig.list("TransportChannelService" ).split("\r\n")[2]
```

   Using Jacl:

```
wsadmin> $AdminConfig list TransportChannelService
(cells/BadgerCell01/nodes/BadgerCellManager01/servers/dmgr|server.xml
#TransportChannelService_1)
(cells/BadgerCell01/nodes/BadgerNode01/servers/nodeagent|server.xml
#TransportChannelService_1095711
814579)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml
#TransportChannelService_109571202
3139)
(cells/BadgerCell01/nodes/BadgerNode01/servers/server2|server.xml
#TransportChannelService_109571203
9302)
wsadmin> set tcs [lindex [$AdminConfig list TransportChannelService] 2]
(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml
#TransportChannelService_109571202
3139)
```

2. Define an outbound TCP channel called testTCPChannel.

   Using Jython:

```
wsadmin>tcpChannel = AdminConfig.create("TCPOutboundChannel", tcs,
[["name", "testTCPChannel"]] )
```

   Using Jacl:

```
wsadmin>set tcpChannel [$AdminConfig create TCPOutboundChannel $tcs
"{name testTCPChannel}"]
testTCPChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
TCPOutboundChannel_1095969213949)
```

3. Define an outbound SSL channel called testSSLChannel. There are two steps required to define such a channel.

   a. Identify the SSL alias to be used by the channel.

      Using Jython:

```
wsadmin>for obj in AdminConfig.list("SSLConfig" ).split("\r\n"):
print obj+AdminConfig.show(obj, "alias")
```

      Using Jacl:

```
wsadmin>foreach obj [$AdminConfig list SSLConfig] { puts "$obj
[$AdminConfig show $obj alias]]" }
(cells/BadgerCell01|security.xml#SSLConfig_1) {alias BadgerCellManager01/
DefaultSSLSettings}]
(cells/BadgerCell01|security.xml#SSLConfig_1095711819776) {alias BadgerNode01/
DefaultSSLSettings}]
```

   b. Create an SSL channel as in the following example, in which the BadgerNode01/ DefaultSSLSettings alias is used.

Using Jython:

```
wsadmin>sslChannel = AdminConfig.create("SSLOutboundChannel", tcs, [["name",
"testSSLChannel"], ["sslConfigAlias","BadgerNode01/DefaultSSLSettings"]])
```

Using Jacl:

```
wsadmin>set sslChannel [$AdminConfig create SSLOutboundChannel $tcs
"{name testSSLChannel}
{sslConfigAlias BadgerNode01/DefaultSSLSettings}"]
testSSLChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
SSLOutboundChannel_1095971760671)
```

4. Define an outbound MQFAP channel called testMQFAPChannel.

   Using Jython:

```
wsadmin>mqfapChannel = AdminConfig.create("MQFAPOutboundChannel", tcs,
[["name", "testMQFAPChannel"]] )
```

   Using Jacl:

```
wsadmin>set mqfapChannel [$AdminConfig create MQFAPOutboundChannel $tcs
"{name testMQFAPChannel}"]
testMQFAPChannel(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#
MQFAPOutboundChannel_1095977512682)
```

5. Finally, create the channel chain by combining the channels defined so far. For example, to create a chain called testChain:

   Using Jython:

```
wsadmin>AdminConfig.create("Chain", tcs, [["name", "testChain"], ["enable",
"true"], ["transportChannels", [tcpChannel, sslChannel, mqfapChannel]]] )
```

   Using Jacl:

```
wsadmin>$AdminConfig create Chain $tcs "{name testChain} {enable true}
{transportChannels {$tcpChannel $sslChannel $mqfapChannel}}"
testChain(cells/BadgerCell01/nodes/BadgerNode01/servers/server1|server.xml#Chain_109
5977640896)
```

Operating buses

Use these tasks to operate service integration buses.

About this task

The topics in this section describe how to display the runtime properties of messaging engines and their associated service integration bus links. The properties are influenced by the following operations:

- Starting and stopping messaging engines.
- Starting and stopping service integration bus links.

Displaying the runtime properties of a messaging engine

Display the runtime properties of a messaging engine using the administrative console.

Before you begin

To be able to retrieve the status of messaging engines, you must be logged into the administrative console with at least monitor authority. If you do not have this authority, the messaging engine status is displayed as ″Unavailable″, even if the messaging engine has started. Also, if you are not logged in with the authority needed to retrieve the status of messaging engines, a SECJ0305I error message is logged in the server's systemOut log file.

About this task

To display the runtime properties of a messaging engine, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus to which your messaging engine belongs.

3. In the content pane, under **Topology**, click **Messaging engines**. A list of messaging engines is displayed.
4. Click the messaging engine name.
5. Click the **Runtime** tab. The status of the messaging engine, that is, whether it is currently started or stopped is displayed.

Displaying the runtime properties of a service integration bus link

Display the runtime properties of a service integration bus link using the administrative console.

About this task

To display the runtime properties of a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine that contains the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link whose properties you want to display.
5. Click the **Runtime** tab. The following properties are displayed:

   Status
   The runtime status of the service integration bus link.

   Activity
   Whether the service integration bus link is currently inactive, active, or its activity is unknown.

Managing messages on message points

Use these tasks to list and act on runtime messages that exist on message points in a service integration bus.

About this task

You can list the message points for bus destinations and messaging engines, and list the messages on a selected message point. You can use the list of messages as part of a troubleshooting task to find messages that need to be deleted.

- “Listing messages on a message point” on page 1250
- “Deleting messages on a message point” on page 1251

Managing service integration buses with administrative commands

You can use these commands to manage service integration buses.

About this task

These commands provide an alternative to using the administrative console or using the more complex syntax of wsadmin and Jython.

The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.

1. Open a wsadmin command session in local mode For example:

   ```
   wsadmin -conntype none -lang jython
   ```

2. Type `AdminTask.`*`command`*, where *command* is the command format as indicated in the related reference topics.

   For example:

```
wsadmin>AdminTask.listSIBusMembers('[-bus bus1 ]')
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092155259869]
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092159844593]
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092160253751]

wsadmin>AdminTask.listSIBEngines('[-bus bus1 ]')
'node01.server1-bus1(cells/cell01/nodes/node01/servers/server1|sib-engines.xml#
SIBMessagingEngine_1212163145962)\r\n
node02.server2-bus2(cells/cell01/nodes/node02/servers/server2|sib-engines.xml#
SIBMessagingEngine_1212163146273)'
```

Messaging engines

These topics provide information about messaging engines, which provide the processing function on a service integration bus.

- Learning about messaging engines
- “Configuring messaging engines” on page 1152
- Operating messaging engines
- “Managing messaging engines with administrative commands” on page 1204
- SIBAdminCommands: Messaging engine administrative commands for the AdminTask object
- Messaging engine troubleshooting tips

Configuring messaging engines

You can configure messaging engines in a variety of ways. For example, you can create and apply security to a messaging engine, then use this engine to send and receive messages. When you add a server cluster to a service integration bus, at least one messaging engine is created automatically. If you also use messaging engine policy assistance, some configuration properties are set automatically.

Configuring messaging engine properties

You can configure the properties of a messaging engine in the administrative console. For example you can select whether the messaging engine is started automatically when its associated application server is started, how many messages it can process, and target groups that the engine can join.

About this task

In most cases, you can configure the properties of a messaging engine without interrupting the processing of messages by the messaging engine.

1. Start the administrative console.
2. Navigate to **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name***.
3. Configure the messaging engine properties. For information about the properties that you can configure, see the property descriptions in Messaging engines [Settings]
4. Click **OK**.
5. Save your changes to the master configuration.

Listing the messaging engines in a bus

You can view the list of existing messaging engines in a bus by using the administrative console. You can decide which messaging engines you want to change, for example which buses they are associated with.

About this task

To list the messaging engines in a bus, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that your messaging engine belongs to.

3. In the content pane, under **Topology**, click **Messaging engines**. The list of messaging engines in the bus is displayed.

Removing a messaging engine from a bus

You can remove a messaging engine from a service integration bus if you no longer require it to send and receive messages on the bus.

Before you begin

You should be wary of deleting and re-creating messaging engines on bus members for which WS-Notification-administered subscribers have been configured, because in some cases this can leave the remote Web service subscription active (and passing notification messages to the local server) even though there is no longer any record of it. For more information, see the WS-Notification troubleshooting tip Deleting administered subscribers and messaging engines.

1. Stop the messaging engine. You can stop either in **Immediate** or **Force** mode, as described in "Stopping a messaging engine" on page 1202.
2. Use the wsadmin command deleteSIBEngine to delete the messaging engine. All service integration bus links, MQ links, and custom properties that are owned by the engine are deleted.

   Note: When you remove a messaging engine, WebSphere Application Server does not delete the data store tables automatically. You must remove them manually, or delete all the rows in all the tables. If you do this, a new messaging engine might fail to start if it attempts to use an orphaned data store. Refer to the documentation for your chosen relational database management system for information about deleting tables.

   Alternatively, for Apache Derby, you can delete the database's directory, which is located in *profile_root*/databases/com.ibm.ws.sib, where *profile_root* is the directory in which profile-specific information is stored. However, do this only if the messaging engine is the sole user of the database.

   For more information, see Data store life cycle.

   Similarly, the file store files are not automatically deleted when you delete the messaging engine. You might want to delete the file store files to reclaim disk space.

Listing the messaging engines defined for a server bus member

You can display a list of messaging engines defined for a server bus member by using the administrative console. You can decide which messaging engines you want to change, for example, which buses they are associated with.

About this task

To display the list of messaging engines, use the administrative console to complete the following steps:

1. In the navigation pane, click either **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** or **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engines**. A list of messaging engines is displayed in the content pane.
2. Optional: Select one or more messaging engines to work with, for example to change the properties of the messaging engine.

Setting up the data store for a messaging engine

To set up a data store for a messaging engine, the database administrator must first create the database. You must then configure the data source and configure the messaging engine to use it.

Before you begin

Before you use this task, review the information in “Planning the configuration of a messaging engine to use a data store” on page 1209, and ensure that your database administrator has taken any appropriate action.

About this task

This task summarizes the steps that you must perform to set up the data store for a messaging engine. You need to work with your database administrator to complete step 1 of this task.

1. Ask your database administrator to perform the following steps:
   a. Create the database. Refer to “Creating the database” on page 1210
   b. Create users and schemas. Refer to “Creating users and schemas in the database” on page 1211
   c. Create the data store tables. Refer to “Creating the tables” on page 1212
   d. Ensure that the user has sufficient privilege to allow the messaging engine to operate. For more information about the privileges that required for the selected database, refer to “Database privileges” on page 1218.
2. When the database administrator has completed step 1, perform the following steps:
   a. Obtain a user ID from your database administrator.
   b. Configure a data source. Refer to “Configuring a JDBC data source for a messaging engine” on page 1213.
   c. Configure the messaging engine to use the data source. Refer to “Configuring a messaging engine data store to use a data source” on page 1154.

Configuring a messaging engine data store to use a data source:

After configuring a data source, you can configure a messaging engine data store to use the data source.

Before you begin

This topic assumes that you have chosen or created a bus and a messaging engine, and that the messaging engine specifies “data store” as its message store type.

You must also have configured a data source, as described in “Setting up the data store for a messaging engine” on page 1153.

About this task

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

Use the WebSphere Application Server administrative console to set the data store configuration parameters.

To configure the messaging engine to use a data source, use the administrative console to complete the following steps:

Service integration → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Message store**

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name***.
2. Check that the **Message store type** is *Data store*.
3. Click **[Additional Properties] Message store**. The data store configuration detail panel is displayed.

4. Specify the following data store configuration details:

Data source JMDI name
Type the JNDI name of the data source that provides access to database that holds the data store.

Schema name
Type the name of the database schema that contains the tables used by the data store.

Note: The schema name is usually the same as the user ID that is declared in the authentication alias. With some databases, for example DB2, you can provide an alternative schema name. For more information about the relationship between users and schema, refer to the documentation for your chosen RDBMS.

Note: When you configure your messaging engine to use an Informix database, you must specify the schema name in lowercase letters.

When starting, a messaging engine that uses a data store checks to see if its data store exists. If the user has selected the **Create tables** option in the configuration, whose tick-box is ticked by default, the messaging engine creates the tables in its chosen schema.

The **Schema name** field is optional, and:

- The default schema name is *IBMWSSIB*.
- If you delete the text so that field is blank, the messaging engine takes the user id defined in the authentication alias to be the schema name.
- If you define a schema name explicitly, that schema name is used by the messaging engine.

Authentication alias
Select the authentication alias that enables access to the data source.

Note: When you create a new Network Attached Apache Derby data store, by default you get a blank authentication alias.

Create tables
Select the check box if you want WebSphere Application Server to create the database tables automatically. For more information, refer to “Creating the tables” on page 1212.

Note: The user ID that the data store uses to connect to the data source must have sufficient authority to create the database tables and indexes.

Note: The option for WebSphere Application Server to create the tables is not available with DB2 for z/OS. Refer to “Enabling your database administrator to create the data store tables” on page 1212 if you use DB2 for z/OS.

Number of permanent tables
Permanent table contains persistent objects for the data store.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Number of temporary tables
Temporary tables nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Configuring service integration bus links

You can configure service integration bus links on messaging engines in a variety of ways. For example, you can start, stop, or remove links.

About this task

When you create a foreign bus connection to connect two service integration buses, a service integration bus link is created automatically. The foreign bus connection contains a routing definition, which is a virtual link, and the service integration bus link is the corresponding physical link on the messaging engine.

Configuring the properties of a service integration bus link:

After establishing a service integration bus link you might want to configure the properties of a service integration bus link such as the name of the service integration bus link, or authentication alias used by foreign bus that the link connects to.

About this task

To configure the properties of a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to configure the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you want to configure.
5. Specify the following properties for the service integration bus link:

   Name The name of the service integration bus link. In order to work, the name must be the same as the name of the corresponding service integration bus link configured on the target foreign bus.

   Description
   An optional description for the service integration bus link, for administrative purposes.

   UUID The universal unique identifier assigned by the system to the service integration bus link for administrative purposes.

   Foreign messaging engine
   The messaging engine on the foreign bus to which this link connects.

   Note: This foreign bus name must not be altered after it has been configured. If you alter it, any messaging engines that already hold state information about the link will not be able to use the link unless the foreign bus name is reset to its original value.

   Target inbound transport chain
   The type of transport chain used for communication with the foreign bus. The transport chain name must be the name of the transport chain as defined on the server on which the target messaging engine is hosted.

   Bootstrap endpoints
   The comma-separated list of endpoints used to connect to a bootstrap server. This property is set in the same way as the **Provider endpoint** property in the JMS connection factory settings. For more information, see steps 1 and 2 of “Configuring a connection to a non-default bootstrap server” on page 1598. Although this task primarily describes how to configure a JMS connection factory, it is also applicable to setting several bootstrap endpoint values if the remote messaging engine is in a cluster.

Note: Service integration bus links over BootstrapTunneledMessaging and BootstrapTunneledSecureMessaging transport chains only work directly between application server instances. Bus links over TunneledMessaging transport chains do not work if an HTTP server is placed in front of either application server instance.

Authentication alias
The name of the authentication alias, used to authenticate access to the foreign bus. The alias must be known to the foreign bus.

Initial state
Whether the link is started automatically when the messaging engine is started. Until started, the gateway link is unavailable. If this property is set to **Started** the service integration bus link is started when the messaging engine is started.

6. Click **OK**.
7. Save your changes to the master configuration.

Listing the service integration bus links:

You can list all the service integration bus links that are linked to a messaging engine.

About this task

To list the service integration bus links for a messaging engine, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that your messaging engine belongs to.
3. In the content pane, under **Additional properties**, click **Messaging engines**. The list of messaging engines in the bus is displayed.
4. In the content pane, select the messaging engine for which you want to list the service integration bus links.
5. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.

Example

The following combinations of **Status** and **Activity** values are possible:

Status	Activity	Meaning
started	inactive	The service integration bus link is started on the local messaging engine but has no connection to the foreign bus. The service integration bus link is attempting to activate a connection to the foreign bus. The service integration bus link on the foreign bus must also be started to successfully activate of a connection between the buses.
started	active	The service integration bus link is started on the local messaging engine and has an active connection to the foreign bus.
stopped	inactive	The service integration bus link is stopped on the local messaging engine and there is no connection to the foreign bus.
unknown	inactive	An error might have occurred in setting up the link, such that the object that is used to report the current state is not available.

What to do next

You can now add or remove a service integration bus link or select a service integration bus link to start, stop, or configure.

Starting a service integration bus link:

When a service integration bus link has been started, it can be used for communicating with its associated messaging engines.

About this task

To start a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine for which you want to start the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to start.
5. Click **Start**.

Stopping a service integration bus link:

When a service integration bus link has been stopped, it cannot be used for communication until it is restarted.

About this task

To stop a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine to which the service integration bus link that you wish to stop belongs.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to stop.
5. Click **Stop**.

Removing a service integration bus link:

When a service integration bus link to a messaging engine has been removed, it cannot be used for communication until it is recreated.

Before you begin

Before you remove the service integration bus link you must stop the link.

When you remove a service integration bus link, all traffic using the link needs to be dealt with in a similar way to when you remove a destination. For further details, see “Deleting a non-topic space bus destination” on page 1246. Note that express Quality of Service (QoS) messages are discarded.

About this task

To remove a service integration bus link from a messaging engine, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select which the messaging engine has own association with the service integration bus link that you wish to remove.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link that you wish to remove.
5. Click **Delete**.
6. Save your changes to the master configuration.

What to do next

When you remove a service integration bus link, all traffic using the link must be diverted by alternative routes or held pending further administrative action. Express QoS messages are discarded.

Starting a messaging engine

You can either start a messaging engine directly by using the administrative console or by starting the server that hosts the messaging engine.

About this task

To start a messaging engine, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that your messaging engine belongs to.
3. Do one of the following to display a list of messaging engines:
   - For a list of messaging engines in the bus, in the content pane, under **Topology**, click **Messaging engines**.
   - For a list of messaging engines for a server, in the content pane, under **Topology**, click **Bus Members**. Click the name of the required server.
   - For a list of messaging engines for a cluster, in the content pane, under **Topology**, click **Bus Members**. Click the name of the required cluster.
4. Select the messaging engine that you wish to start.
5. Click **Start**.

Stopping a messaging engine

You can either stop a messaging engine directly or stop it by stopping the server that hosts the messaging engine.

About this task

A messaging engine can stop in two ways:

Immediate
The messaging engine is stopped on completion of all messaging operations that are being carried out at the time of the stop request.

Force The messaging engine is stopped without allowing messaging operations to complete and applications are forcefully disconnected.

To stop a messaging engine, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that the messaging engine belongs to.
3. Do one of the following to display a list of messaging engines:

- For a list of messaging engines in the bus, in the content pane, under **Topology**, click **Messaging engines**.
- For a list of messaging engines for a server, in the content pane, under **Topology**, click **Bus Members**. Click the name of the required server.
- For a list of messaging engines for a cluster, in the content pane, under **Topology**, click **Bus Members**. Click the name of the required cluster.

4. Select the messaging engine that you wish to stop.
5. Select **Immediate** or **Force** from the **Stop mode** drop-down list.
6. Click **Stop**.

What to do next

Note: If an immediate stop is taking too long, you can escalate it to a force stop by selecting the **Force** option. This overrides your previous selection of the **Immediate** option.

Displaying the runtime properties of a messaging engine

Display the runtime properties of a messaging engine using the administrative console.

Before you begin

To be able to retrieve the status of messaging engines, you must be logged into the administrative console with at least monitor authority. If you do not have this authority, the messaging engine status is displayed as ″Unavailable″, even if the messaging engine has started. Also, if you are not logged in with the authority needed to retrieve the status of messaging engines, a SECJ0305I error message is logged in the server's systemOut log file.

About this task

To display the runtime properties of a messaging engine, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus to which your messaging engine belongs.
3. In the content pane, under **Topology**, click **Messaging engines**. A list of messaging engines is displayed.
4. Click the messaging engine name.
5. Click the **Runtime** tab. The status of the messaging engine, that is, whether it is currently started or stopped is displayed.

Displaying the runtime properties of a service integration bus link

Display the runtime properties of a service integration bus link using the administrative console.

About this task

To display the runtime properties of a service integration bus link, use the administrative console to complete the following steps:

1. Display the list of messaging engines.
2. In the content pane, select the messaging engine that contains the service integration bus link.
3. In the content pane, under **Additional properties**, click **Service integration bus links**. A list of service integration bus links is displayed.
4. Select the service integration bus link whose properties you want to display.
5. Click the **Runtime** tab. The following properties are displayed:

Status
The runtime status of the service integration bus link.

Activity
Whether the service integration bus link is currently inactive, active, or its activity is unknown.

Managing messaging engines with administrative commands

You can use these commands to manage messaging engines.

About this task

These commands provide an alternative to using the administrative console or using the more complex syntax of wsadmin and Jython.

1. Open a wsadmin command session in local mode. For example:

```
wsadmin -conntype none -lang jython
```

   The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.

2. Type `AdminTask.`*`command`*, where *command* is the command format as indicated in the related reference topics.

   For example:

```
wsadmin>AdminTask.listSIBusMembers('[-bus bus1 ]')
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092155259869]
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092159844593]
[cells/cell01/buses/bus1|sib-bus.xml#SIBusMember_1092160253751]

wsadmin>AdminTask.listSIBEngines('[-bus bus1 ]')
'node01.server1-bus1(cells/cell01/nodes/node01/servers/server1|sib-engines.xml#
SIBMessagingEngine_1212163145962)\r\n
node02.server2-bus2(cells/cell01/nodes/node02/servers/server2|sib-engines.xml#
SIBMessagingEngine_1212163146273)'
```

Message stores

Message stores play an essential part in the operation of messaging engines. Each messaging engine has one and only one message store. This can either be a file store or a data store.

About this task

A message store enables a messaging engine to preserve operating information and to retain those objects that messaging engines need for recovery in the event of a failure.

A messaging engine preserves both volatile and durable data in its message store. Volatile data is lost when a messaging engine stops, in either a controlled or an uncontrolled manner. Durable data is available after the server restarts. For more information, refer to Message reliability levels. A messaging engine stores various types of data, including messages, transaction states, and communication channel states.

When started, a messaging engine obtains configuration information from the WCCM (WebSphere Application Server Common Configuration Model) repository. A messaging engine retrieves all other data from its own file store or data store.

Note: There are currently no facilities available for migrating from a data store to a file store.

Considerations when choosing between a file store and a data store

Choosing to use a file store for your messaging engine can have several advantages over using a data store.

1. Better performance

To achieve best performance using a data store, you often need to use a separate remote database server. File store can exceed the performance of a data store using a remote database server without needing a separate server.

2. Low administration requirements

   The file store combines high throughput with little or no administration. This makes it suitable for those who do not want to worry about where the messaging engine is storing its recoverable data. File store improves on the throughput, scalability, and resilience of Apache Derby.

3. Lower deployment costs

   Use of data store might require database administration to configure and manage your messaging engines. File store can be used in environments without a database server.

As a managerial decision, some organizations prefer to use data store because it utilizes their existing resources more effectively. For example, this might be the case for a company with a strong team of database specialists, or a stable database infrastructure.

One technical advantage of using data store is that some Java EE applications can share JDBC connections to benefit from one-phase commit optimization. For more information refer to Sharing connections to benefit from one-phase commit optimization. File store does not support this optimization.

Data stored in both data store and file store benefit from security features provided by WebSphere Application Server when accessed using the WebSphere Application Server APIs, that is JMS messaging. Further security features can be taken advantage of depending on the type of message store you use. For more details see File stores and Data stores.

- Data store: access to your chosen database will be through a userid/password that will be administered using the supplied tools for your specified DBMS. Logical and physical separation of your database server can also be used to improve the overall security of your data.
- File store: additional security can be provided when using a file store by careful consideration of your file store files. For example making use of a secure network attached drive to store your file store files will improve the physical security of your data. Another example would be using an operating system encrypted file system to store your files on.

Both file store and data store offer high availability capabilities, for more details refer to “Message store high availability considerations.”

Message store high availability considerations

High availability refers to the capability of failing over messaging engines between servers. Both file stores and data stores can be deployed to be used in a highly available environment.

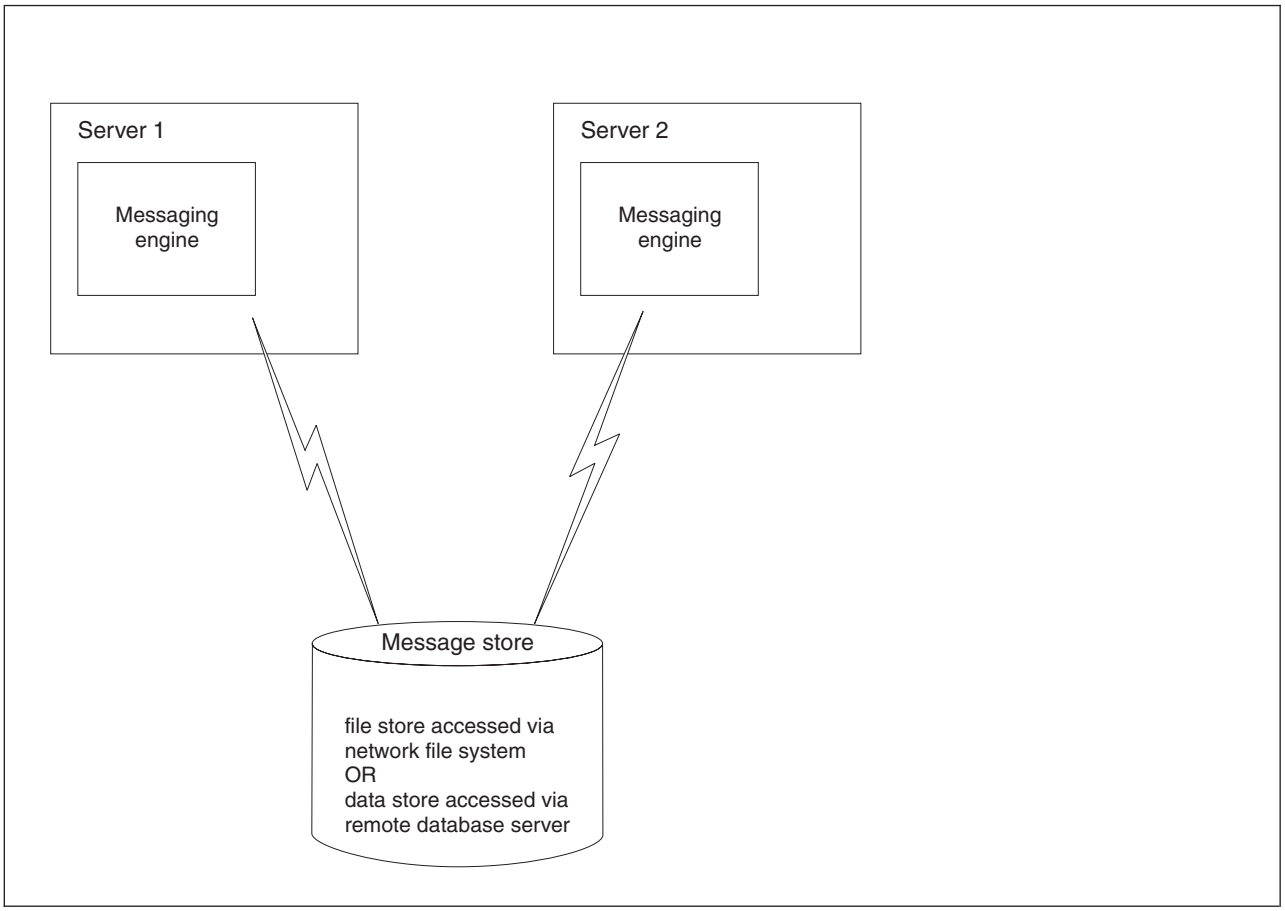


Figure 15. Failover of a messaging engine between servers.

For more information see related links.

Managing file stores

A file store is a message store which directly uses files in the file system via the operating system. Each messaging engine has one and only one file store.

Configuring a messaging engine to use a file store

When a messaging engine is created, it uses a file store by default. You can configure the file store attributes to suit system needs.

About this task

You set up a messaging engine to use a file store by accepting the default choice of message store when you are creating a bus and adding a new bus member. You can choose either to accept the default settings that the administrative console displays or to make changes to these settings. After the file store has been created, you can, if required, subsequently make changes to the current file store settings.

Modifying file store configuration:

After a file store has been created, you can, if required, modify file store attributes to suit your system needs.

About this task

When you add a new bus member that uses a file store, you can choose either to accept the default settings for the file store or make changes to these settings, depending on your requirements.

After the file store has been created, you can, if required, subsequently modify the log size, permanent store size or temporary store size settings directly through the administrative console. The new values take effect the next time that the messaging engine is started.

The directory paths for the file store are fixed when the file store is created, and cannot subsequently be modified. You can create a new file store in a new location. To do this you remove the server or cluster as a bus member and then add it again choosing a new location for the file store. Any messages remaining in the original file store would be lost during this process.

Note: The default configuration for a file store is intended to be sufficient when used in typical messaging workloads. To improve the performance or resilience of the log file, the permanent store file or the temporary store file, you can modify the file store attributes to control where these files are placed. For example, you can achieve better performance if you place these three file store files in a faster disk. Similarly, you can control the sizes of the log file, the permanent store file and the temporary store file to handle workloads with a large number of active transactions, large messages or a large volume of message data resident in the messaging engine.

- To make changes to the log size, permanent store size or temporary store size, complete the following steps:
  1. Open the Administrative console.
  2. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Message store**.
  3. Make the changes that you require to the existing log size, permanent store size or temporary store settings, then click **OK**. For more information about configuring the properties, see File store [Settings]
  4. Save your changes to the master configuration.
- To choose a new location for a file store, complete the following steps:
  1. To avoid losing messages on destinations that are localized on the file store, first ensure that no messages are left on the queues.
  2. Remove the server or cluster as a member of the bus.
  3. Add the server and cluster as a bus member again, choosing a new location for the file store.

Deleting files following removal of a messaging engine:

Removing a messaging engine from the configuration does not automatically delete the file store's files. You need to also find and delete the log file, the permanent store file and the temporary store file from the disk in order to reclaim disk space.

About this task

Note: Files comprising a file store still remain when the messaging engine is deleted. If you are using the default values for the file store directory names, it is possible to delete and then recreate a messaging engine with the same name without manually removing the files. This is due to the presence of the messaging engine's Universal Unique IDentifier (UUID) in the default log and store directory names of a file store.

Example: If the deleted messaging engine is called *messagingengine0*, and the new one *messagingengine1* then the log files in the old and new file stores are

```
C:\{USER_INSTALL_ROOT}\filestores\..\messagingengine0\..\logfile
```

and

```
C:\{USER_INSTALL_ROOT}\filestores\..\messagingengine1\..\logfile
```

The two log files coexist in the same file stores directory. You do not have to delete the old files if you choose.

1. Locate the directory in which your old file store log files and store files are stored.
2. Delete the redundant files using the facilities of the operating systems.

Results

After deleting the files left by the redundant file store, free disk space becomes available to your file system.

Administering messaging engine file stores

A file store is a type of message store that directly uses files in a file system through the operating system. Therefore, administration of the file store depends on the type of operating system.

About this task

- “Backing up a file store”
- “Restoring a file store”

Backing up a file store:

Backing up the files in a file store should be done according to your operating system or by using a backup tool.

About this task

A file store is a type of message store that directly uses files in a file system through the operating system. Use the facilities of your operating system or a backup tool to take a backup copy of the log file, the permanent store file and the temporary store file, which comprise a file store. For more information about these files, see File store configuration attributes.

Note: You must not start backing up files while the messaging engine is still running or data might be corrupted.

Note: You must treat the log file, the temporary store file and the permanent store file as one unit, that is, any operations must be performed on all three files.

Restoring a file store:

Restoring the files in a file store should be done according to your operating system or by using a backup tool to restore the backup copies of your files.

Before you begin

Before you start this task, make sure that the messaging engine is not running.

Note: You must not restore a file store while the messaging engine is still running or data might be corrupted.

About this task

When performing this task, you must treat the log file, the temporary store file and the permanent store file which comprise a file store as one unit, that is, any operations must be performed on all three files. For more information about these files, see File store configuration attributes.

Use the facilities of your operating system or a backup tool to restore the backup copy of the log file, the permanent store file and the temporary store file.

Managing data stores

A data store is a message store that uses a relational database. A data store consists of a set of tables that are in the same database schema. It is used by a messaging engine to store operating information in the database, as well as to preserve essential objects that the messaging engine needs for recovery in the event of a failure.

About this task

A data store consists of the set of tables that a messaging engine uses to store persistent data in a database. Refer to “Data store tables” on page 1217 for a list of the tables that comprise a data store. All the tables in a data store are held in the same database schema. You can create multiple data stores in the same database only by using a different schema for each data store.

Configuring a messaging engine to use a data store

Although the default message store for a typical messaging engine is file store you can also configure a messaging engine to use a data store.

About this task

Each messaging engine has its own file store or data store. If the data store is chosen the messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

When a new messaging engine that uses a data store is created on a single server, it is configured to use an Apache Derby data source by default. This enables the messaging engine to run without needing any additional configuration.

If you want to configure a new messaging engine to use your chosen data source when you create that messaging engine on a single server, refer to “Adding a server as a new bus member” on page 1145.

If you do not want to use the default data source configuration, you can use the WebSphere Application Server administrative console to change the configuration parameters. For example, you can change the data source or you can configure the data store to use a different JDBC provider.

Planning the configuration of a messaging engine to use a data store:

There are a number of choices that you need to consider before you configure a messaging engine to use a data store.

1. Choose the relational database management system (RDBMS) for the data store. You might want to choose the RDBMS that you use for other applications, particularly if you are already familiar with the tools you use for managing that RDBMS. You might also want to consider the following criteria:
   - Performance
   - Scalability
   - Availability, especially if you are running messaging engines in a high availability environment

   When a new messaging engine that uses a data store is created on a single server, it is configured to use an Apache Derby data source by default. This enables the messaging engine to run without needing any additional configuration. The default embedded Derby data source is sufficient for many purposes. Other relational database management systems offer more comprehensive tooling and improved performance, particularly scalability on larger machines with more than two processors.

2. Choose your database topology. In some cases, running the data store on a remote node can improve performance. In other cases, a local database provides performance equivalent to a remote database. You might want to conduct your own performance analysis, because the performance characteristics can be very sensitive to the hardware specification.
3. Consider whether you want WebSphere Application Server to create the data store tables automatically or whether you want your database administrator to create the tables beforehand. For more information, refer to “Creating the tables” on page 1212.
4. Message data is stored in a database table column of datatype BLOB. Before you create a datastore, you must consider the size of your expected workload to ensure that your database administrator creates a sufficiently large enough BLOB space to hold your message data.

Selecting the data store topology:

You have several options to consider when selecting the relative location of a data store and its messaging engine.

- Decide whether the data store will run on the same node as its messaging engine or on a remote node.
- Decide whether the data store will have a dedicated database or share a database with other data stores.

Setting up the data store for a messaging engine:

To set up a data store for a messaging engine, the database administrator must first create the database. You must then configure the data source and configure the messaging engine to use it.

Before you begin

Before you use this task, review the information in “Planning the configuration of a messaging engine to use a data store” on page 1209, and ensure that your database administrator has taken any appropriate action.

About this task

This task summarizes the steps that you must perform to set up the data store for a messaging engine. You need to work with your database administrator to complete step 1 of this task.

1. Ask your database administrator to perform the following steps:
   a. Create the database. Refer to “Creating the database”
   b. Create users and schemas. Refer to “Creating users and schemas in the database” on page 1211
   c. Create the data store tables. Refer to “Creating the tables” on page 1212
   d. Ensure that the user has sufficient privilege to allow the messaging engine to operate. For more information about the privileges that required for the selected database, refer to “Database privileges” on page 1218.
2. When the database administrator has completed step 1, perform the following steps:
   a. Obtain a user ID from your database administrator.
   b. Configure a data source. Refer to “Configuring a JDBC data source for a messaging engine” on page 1213.
   c. Configure the messaging engine to use the data source. Refer to “Configuring a messaging engine data store to use a data source” on page 1154.

Creating the database:

When creating the database for your messaging engine using a data store you must do it according to your Relational Database Management System (RDBMS).

Before you begin

Choose which Relational Database Management System (RDBMS) you want to use for the data store. Unless you are using the embedded Apache Derby provider, create the database before you create a messaging engine. Make a note of the database parameters that you need for configuring the data source. Refer to “Configuring a JDBC data source for a messaging engine” on page 1213 for more information.

Refer to the documentation for your chosen RDBMS for information about how to create a database. The default database for a data store is an embedded Apache Derby database. If you have chosen to configure the bus member to use a data store with default settings, it can only be a server. Unless the data store database exists already, the messaging engine creates the database automatically when the messaging engine makes its initial connection.

Note:

- Ensure that you create the database server with a page size of at least 4k.
- Ensure that you set the **lock scheme** property on your server to the value *datarows*. This avoids the possibility of a deadlock on the data store tables.
- Ensure that you set the **enable housekeeper GC** property on your server to the value *5*. This improves the ability of the server to reclaim redundant space within your database when it is under heavy load.
- Ensure that you select the **allow nulls by default** option for your database instance. This is required for the correct operation of the messaging engine.

Note: The one-to-one relationship between a messaging engine and a data store means that every messaging engine must have its own database tables. If you are using the Informix RDBMS, configure a separate database instance for each messaging engine. Problems have been observed in this environment when the data stores for multiple messaging engines were configured to use separate schemas in the same database.

Creating users and schemas in the database:

After you have created a database, you need to create the schema in which all tables in the data store are held.

Before you begin

Before you begin this task, create the database for your messaging engine.

About this task

All the tables in the data store must be stored in the same schema. If the schema names are different, a database can hold more than one data store. Although every messaging engine uses the same table names, its relationship with the schema gives each messaging engine exclusive use of its own tables.

To connect to WebSphere Application Server, the database administrator must create at least one user. The number of user IDs you need depends on the database you use:

- If you are using Derby, DB2 or Oracle, then the messaging engine can be configured to create any additional schemas that may be required for other data stores. That is, if you only create one user, it can have one to many relationships with the schemas in the database. Refer to “Configuring a messaging engine data store to use a data source” on page 1154 for details.
- For all other types of databases, the schema must be created prior to starting the messaging engines that depend on them.

If the user ID can be configured to use multiple schemas, then only that user ID is needed for all messaging engines. Otherwise the user is restricted to using tables in its own schema. In this case there can only be one user ID per schema.

1. Refer to the documentation for your chosen relational database management system (RDBMS) for more information about how to create users and schema. With DB2 databases, you create users and schema in separate steps. With the other databases, there is a one-to-one relationship between a schema and a user.
2. Refer to “Database privileges” on page 1218 for information about the authorities that you require to access the data store.

Creating the tables:

When creating the data store tables, you have two options. You can either choose that the database administrator does it manually, or that WebSphere Application Server does it automatically.

Before you begin

Decide whether you want your database administrator to create the tables, or WebSphere Application Server to create the tables automatically.

- If you want your database administrator to create the database tables, refer to “Enabling your database administrator to create the data store tables.”
- If you want WebSphere Application Server to create the tables, refer to “Enabling WebSphere Application Server to create the data store tables”

Enabling WebSphere Application Server to create the data store tables:

WebSphere Application Server can create the tables for the messaging engine data store automatically.

Before you begin

Before you use this task, decide whether you want WebSphere Application Server to create the tables.

Note: The option for WebSphere Application Server to create the tables is not available with DB2 for z/OS. Refer to “Enabling your database administrator to create the data store tables” if you use DB2 for z/OS.

1. Ensure that WebSphere Application Server has sufficient authority to create tables and indexes. For more information about the privileges that you require for your chosen database, refer to “Database privileges” on page 1218.
2. When you configure your messaging engine data store, ensure that **Create tables** is selected. For more information, refer to “Configuring a messaging engine data store to use a data source” on page 1154.

   Note: Do not select **Create tables**, otherwise an exception will be thrown when WebSphere Application Server attempts to create the tables.

Enabling your database administrator to create the data store tables:

Before you begin

Before you use this task, review the information in “Planning the configuration of a messaging engine to use a data store” on page 1209, and ensure that your database administrator has taken any appropriate action.

About this task

Use the sibDDLGenerator command to generate the DDL statements that the database administrator will need to create the tables for the messaging engine data store:

1. At a command prompt, issue the sibDDLGenerator command and redirect the output to a file. Refer to sibDDLGenerator command for a description of the sibDDLGenerator command.

   Note: If you want to process the DDL statements with a command line processor that requires the statements to conform to a specific format, use the optional parameters that control the format of the DDL statements. For example, if each statement must be terminated with a semicolon, use **-statementend** *;*

   To access the i5/OS command line, or run an i5/OS command line program, use the STRQSH command to launch a Qshell session. For more information, see Configuring Qshell to run WebSphere Application Server scripts .
2. Send the output file to your database administrator to process the DDL statements that are generated. The DDL statements can be ported across operating systems, for example you can generate the DDL statements on a machine running the Windows operating system and then run them on a machine running the z/OS operating system.

   Note:

   - Your database administrator can modify the DDL statements, but *must not* modify the table names or the column names in any way. Doing so might cause the messaging engine to fail to start.
   - If the DDL statements are to be run on the z/OS operating system, your database administrator must change the VCAT name in the first line of the DDL statements (the create storage group statement) to a valid high level qualifier for their system.

Configuring a JDBC data source for a messaging engine:

Learn about how to use the WebSphere Application Server administrative console to configure a data source for a messaging engine.

Before you begin

Note:

When configuring a service integration bus member to use a data store, be aware that using a type 2 JDBC driver for the data store is not supported for configurations where WebSphere MQ server definitions are also used. If your configuration includes WebSphere MQ server definitions and you are using a data store you must use type 4 JDBC drivers.

Note: When you create a new Network Attached Apache Derby data store, by default you get a blank authentication alias.

Note: Use the Oracle 10g thin driver for the service integration data store. This driver is compatible with earlier versions of Oracle.

About this task

Each messaging engine has its own file store or data store. If the data store is chosen the messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

Use the WebSphere Application Server administrative console to set the data source configuration parameters. Note that your choice of relational database management system (RDBMS) determines the parameters you set.

Perform the following steps to configure the data source:

1. Create a JDBC provider. Refer to Creating and configuring a JDBC provider using the administrative console. Under **General Properties**, in the **Select the implementation type** field, ensure that you select *Connection pool data source*.

   For information about the settings for your chosen RDBMS, refer to Vendor-specific data sources minimum required settings.
2. Create a data source for the JDBC provider. Refer to Creating and configuring a data source by using the administrative console.
   a. Under **Additional Properties**, ensure that you select *Data sources*.
   b. Configure the connection pool for the JDBC data source. Set the **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10. For more information, refer to Tuning the JDBC data source of a messaging engine.
3. Test the connection. Refer to Test connection.

Configuring a messaging engine data store to use a data source:

After configuring a data source, you can configure a messaging engine data store to use the data source.

Before you begin

This topic assumes that you have chosen or created a bus and a messaging engine, and that the messaging engine specifies “data store” as its message store type.

You must also have configured a data source, as described in “Setting up the data store for a messaging engine” on page 1153.

About this task

A messaging engine uses an instance of a JDBC data source to interact with the database that contains the data store for that messaging engine.

Use the WebSphere Application Server administrative console to set the data store configuration parameters.

To configure the messaging engine to use a data source, use the administrative console to complete the following steps:

Service integration → Buses → *bus_name* → [Topology] Messaging engines → *engine_name* → [Additional Properties] Message store

1. In the navigation pane, click **Service integration → Buses → *bus_name* → [Topology] Messaging engines → *engine_name***.
2. Check that the **Message store type** is *Data store*.
3. Click **[Additional Properties] Message store**. The data store configuration detail panel is displayed.
4. Specify the following data store configuration details:

   Data source JMDI name
   Type the JNDI name of the data source that provides access to database that holds the data store.

   Schema name
   Type the name of the database schema that contains the tables used by the data store.

Note: The schema name is usually the same as the user ID that is declared in the authentication alias. With some databases, for example DB2, you can provide an alternative schema name. For more information about the relationship between users and schema, refer to the documentation for your chosen RDBMS.

Note: When you configure your messaging engine to use an Informix database, you must specify the schema name in lowercase letters.

When starting, a messaging engine that uses a data store checks to see if its data store exists. If the user has selected the **Create tables** option in the configuration, whose tick-box is ticked by default, the messaging engine creates the tables in its chosen schema.

The **Schema name** field is optional, and:

- The default schema name is *IBMWSSIB*.
- If you delete the text so that field is blank, the messaging engine takes the user id defined in the authentication alias to be the schema name.
- If you define a schema name explicitly, that schema name is used by the messaging engine.

Authentication alias
Select the authentication alias that enables access to the data source.

Note: When you create a new Network Attached Apache Derby data store, by default you get a blank authentication alias.

Create tables
Select the check box if you want WebSphere Application Server to create the database tables automatically. For more information, refer to “Creating the tables” on page 1212.

Note: The user ID that the data store uses to connect to the data source must have sufficient authority to create the database tables and indexes.

Note: The option for WebSphere Application Server to create the tables is not available with DB2 for z/OS. Refer to “Enabling your database administrator to create the data store tables” on page 1212 if you use DB2 for z/OS.

Number of permanent tables
Permanent table contains persistent objects for the data store.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Number of temporary tables
Temporary tables nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement.

Note: You can only increase the number of permanent tables or temporary tables, not decrease them.
See related links for more information.

Administering messaging engine data store
Administering the data store for a messaging engine involves restoration and back up of the data store.

About this task

For more information about administering messaging engine data store, see the following topics:
- “Backing up a data store” on page 1216
- “Restoring a data store” on page 1216

Backing up a data store:

Learn to back up a data store

About this task

A data store consists of the set of tables that a messaging engine uses to store persistent data in a database. Refer to “Data store tables” on page 1217 for a list of the tables that comprise a data store. All the tables in a data store are held in the same database schema. You can create multiple data stores in the same database only by using a different schema for each data store.

To back up this data, refer to the documentation for your chosen database.

Backup can make use of the suspended I/O feature of DB2 if that RDBMS is used as the persistent store. With other databases that do not possess this capability, the backup could present a longer interruption to service, or require that the messaging engine is stopped while the backup is made. If you attempt to back up the data store for a messaging engine that is still running, you might lose or corrupt important data.

Include the tables that the topic “Data store tables” on page 1217 describes.

Restoring a data store:

Learn to restore a data store

About this task

A data store consists of the set of tables that a messaging engine uses to store persistent data in a database. Refer to “Data store tables” on page 1217 for a list of the tables that comprise a data store. All the tables in a data store are held in the same database schema. You can create multiple data stores in the same database only by using a different schema for each data store.

To restore this data, refer to the documentation for your chosen database.

Note: You must stop the messaging engine before restoring the data store. If you attempt to restore the data store for a messaging engine that is still running, you might lose or corrupt important data.

Include the tables that the topic “Data store tables” on page 1217 describes.

Increasing the number of item tables for a messaging engine when tables are not automatically created

If a concurrency bottleneck had occurred on the item tables, increasing the number of them will in turn increase the throughput of the messaging engine.

Before you begin

Before performing this task you must ensure that the messaging engine is using a data store, and its *Create tables automatically* option is set to *False*. The purpose of this task is to relieve concurrency bottleneck, which is documented in Increasing the number of data store tables to relieve concurrency bottleneck. For more information on the creation of tables consult “Creating the tables” on page 1212.

1. Relevant performance monitoring tools show that the throughput of a messaging engine is inefficient.
2. Use your database performance monitoring tools to examine lock statistics for the item tables for evidence of a bottleneck. Consult database documentation on how to interpret the locking statistics.
3. Create tables and increase data store attributes.
   - Create tables for the data store’s schema. For more information see “Creating the tables” on page 1212

- Increase the data store's number of permanent tables or temporary tables, or both. For more information see "Configuring a messaging engine data store to use a data source" on page 1154

You can only increase the number of permanent tables or temporary tables, not decrease them.

4. Stop and restart the WebSphere Application Server so that configuration changes take effect. The extra tables are used when the messaging engine restarts.
5. Observe the effect on throughput and lock statistics by checking performance monitoring tools. Consider whether any improvement is sufficient and modifying the data store attributes further would be beneficial

Increasing the number of item tables for a messaging engine when tables are automatically created

If a concurrency bottleneck had occurred on the item tables, increasing the number of them will in turn increase the throughput of the messaging engine.

Before you begin

This task only applies if the messaging engine is using a data store, and its *Create tables automatically* option is set to *True*. The purpose of this task is to relieve concurrency bottleneck, which is documented in Increasing the number of data store tables to relieve concurrency bottleneck. For more information on the creation of tables consult "Creating the tables" on page 1212.

1. Relevant performance monitoring tools show that the throughput of a messaging engine is insufficient.
2. Use your database performance monitoring tools to examine lock statistics for the item tables for evidence of a bottleneck. Consult your database documentation on how to interpret the locking statistics.
3. Increase the attributes for the messaging engine's data store: the number of permanent tables or temporary tables, or both. For more information see "Configuring a messaging engine data store to use a data source" on page 1154. You can only increase the number of permanent tables or temporary tables, not decrease them.
4. Stop and restart the messaging engine so that configuration changes take effect. The extra tables are created when the messaging engine starts again.
5. Observe the effect on throughput and lock statistics by checking performance monitoring tools. Consider whether any improvement is sufficient and modifying the data store attributes further would be beneficial

Data store tables

A data store consists of the set of tables that a messaging engine uses to store data persisted in a database. These data are important when backing up or restoring a data store. Users can spread the data store across multiple tables.

Table name	Purpose
SIBOWNER	Ensures exclusive access to the data store by an active messaging engine
SIBOWNERO	Used for locking the data store This table stores no data in its one EMPTY_COLUMN column
SIBCLASSMAP	Catalogs the different object types in the data store
SIBLISTING	Catalogs the SIBnnn tables
SIBXACTS	Maintains the status of active two-phase commit transactions
SIBKEYS	Assigns unique identifiers to objects in the messaging engine
SIBnnn, where nnn is a number	Contains persisted objects such as messages and subscription information These tables hold both persistent and nonpersistent objects, using separate tables for the different types of data.

Note: The SIBOWNERO table is new for WebSphere Application Server Version 7.0 and must be created when you are migrating from an earlier version of WebSphere Application Server. See Migrating a messaging engine based on a data store for things to consider when you are migrating a messaging engine based on a data store.

Database privileges

In order for a messaging engine to use its data store, the database user ID that the messaging engine uses must have sufficient privilege to access the data store tables.

The following table describes the database privileges that the messaging engine user requires to access the data store.

Database management system	Minimum privilege required for the messaging engine to use the data store tables	Additional privilege required for the messaging engine to create the data store tables
DB2	The messaging engine user ID needs **SELECT**, **INSERT**, **UPDATE**, and **DELETE** privileges on the tables	The messaging engine user ID needs **CREATETAB** authority on the database and **USE** privilege on the table space as well as **CREATEIN** privilege on the schema.
Oracle	The messaging engine user ID needs at least **SESSION** privilege to connect to the database. If the same user ID owns both the data store schema and the messaging engine that is connecting to the database, the messaging engine has sufficient privilege to manipulate the tables. Otherwise, the messaging engine needs **SELECT**, **INSERT**, **UPDATE** and **DELETE** object privileges on the tables that comprise the data store, and **DROP ANY TABLE** system privilege to enable use of the TRUNCATE TABLE statement.	The messaging engine user ID requires sufficient privilege to create relational tables and indexes in the data store schema. The messaging engine also requires a space quota in the default tablespace of the owner of that schema.
SQL Server	Configure the SQL Server for **SQL Server and Windows** authentication. This allows authentication to be based on an SQL server login ID and password. The messaging engine user ID can be the owner of the tables, or be a member of a group that has sufficient authority to issue TRUNCATE TABLE statements.	The messaging engine user ID needs **CREATE TABLE** statement privilege.
Sybase	The messaging engine user ID can be the owner of the tables, or be a member of a group that has sufficient authority to issue TRUNCATE TABLE statements.	The messaging engine user ID needs **CREATE TABLE** permission.
Informix	The messaging engine user ID must have **CONNECT** privilege on the database. It must also have **SELECT**, **INSERT**, **UPDATE** and **DELETE** authority on the tables.	The messaging engine user ID must have **RESOURCE** privilege on the database.
Derby	If user authentication is enabled, you must authorize the messaging engine user ID to access the database. **Note:** The default database that is generated by the messaging engine has no security mechanisms enabled.	You need no additional privileges.

Avoiding message store errors when creating a messaging engine

Using different combinations of parameters can create a file store or a data store according your requirements. The outcome varies in server and cluster scopes.

Server scope

When you add a server as a new bus member take note of the following:

- If you do not specify the type of message store, then a file store is created by default. If you set **Create default data source** to *True* or supply a **Data source JNDI name**, then a data store is created.
- If you choose to use a **file store**, then only file store attributes are presented to be specified. For example the **log file directory**.
- If you choose to use a **data store**:
  - Only data store attributes are presented to be specified. For example the **Data source JNDI name**.
  - If you set **Create default data source** to *False* then you must specify a **Data source JNDI name**.

For more information about file store and data store refer to “Message stores” on page 1204.

Cluster scope

When you add a cluster as a new bus member take note of the following:

- If you choose to use a **file store**:
  - You must not use the default log file, permanent store file and temporary store file directories because they are not suitable for cluster engines.
  - You must specify the **Log directory**, **Permanent store directory** and **Temporary store directory** to be at locations that all members of the cluster can access on your file system.
- If you choose to use a **data store**:
  - You must not use the default data source because it is not suitable for cluster engines.
  - You must specify the **Data source JNDI name** of an existing data source.

For more information about file store and data store refer to “Message stores” on page 1204.

Avoiding errors when creating a messaging engine with a file store or a data store by using the wsadmin tool

Using different combinations of parameters can create a file store or a data store according your requirements. The outcome varies in server and cluster scopes.

Server scope

When you add a server as a new bus member using administrative commands (by specifying `createSIBEngine -server`) take note of the following:

- If you do not specify the type of message store, then a file store is created by default. If you specify `-createDefaultDatasource` or `-dataSourceJndiName`, then a data store is created.
- If you choose to use a **file store** (by specifying `-filestore`), then only file store attributes can be specified.
- If you choose to use a **data store** (by specifying `-datastore`):
  - Only data store attributes can be specified.
  - If you set `-createDefaultDatasource` to `False` then you must specify `-dataSourceJndiName`.

Cluster scope

When you add a cluster as a new bus member using administrative commands (by specifying `createSIBEngine -cluster`) take note of the following:

- If you choose to use a **file store** (by specifying `-filestore`):
  - You must not use the default log file, permanent store file and temporary store file directories because they are not suitable for cluster engines.
  - You must specify `-logDirectory`, `-permanentStoreDirectory` and `-temporaryStoreDirectory` to be at locations that all members of the cluster can access on your file system.
- If you choose to use a **data store** (by specifying `-datastore`):
  - You must not specify `-createDefaultDatasource` because the default data source is not suitable for cluster engines.
  - You must specify `-dataSourceJndiName`, giving the name of an existing data source.

Bus destinations

This topic is the entry-point into a set of topics about bus destinations within a service integration bus, to which applications can attach as producers, consumers, or both to exchange messages.

- Learning about bus destinations
- “Configuring bus destinations” on page 1159
- “Configuring message points” on page 1248
- “Managing messages on message points” on page 1194
- “Administering durable subscriptions” on page 1251

Configuring bus destinations

Use the following tasks to configure permanent bus destinations on service integration buses.

About this task

The steps involved in configuring a bus destination depend on the intended usage of the destination.

For example, the following figure shows a basic scenario based on an application using a JMS queue for point-to-point messaging.

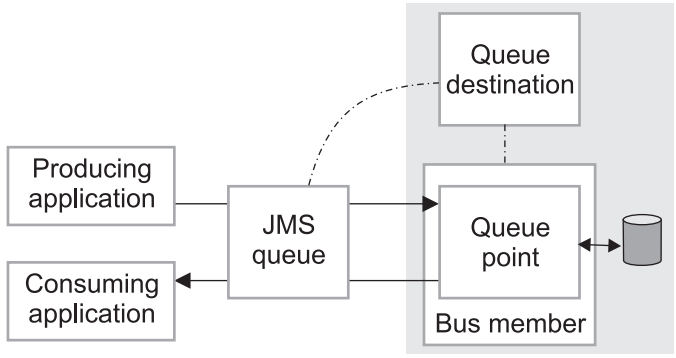


Figure 16. Application use of destinations - basic case

To enable the JMS applications to use a queue destination, you configure the following administrative destination objects:

1. A queue destination on a service integration bus. This configures the properties of the queue, such as the name, and associates the queue with one bus member (an application server). This also automatically configures, on the bus member, a queue point where messages for the queue are held and processed.

2. A JMS queue, which configures the name that applications can use to look up the queue in JNDI. The JMS queue encapsulates the name of the queue destination, as defined in the queue destination above, together with other properties to be used by applications.

After you have created a queue destination, you can optionally configure the queue point to override some properties configured on the queue destination. You can also perform other configuration tasks on the destination and its queue point, and can act on the runtime view.

Each messaging engine has a default exception destination, named `_SYSTEM.Exception.Destinaton.`*`me_name`*. This exception destination can be used to handle messages that cannot be delivered for all bus destination that are localized to the messaging engine. Each bus destination can be configured with a non-default exception destination.

For more information about configuring bus destinations, see the following topics:
- Listing bus destinations
- Creating a bus destination
- Configuring qualities of service for a destination
- Deleting a non-topicspace bus destination
- Deleting a topicspace

Listing bus destinations

Use this task to display administrative lists of bus destinations for a service integration bus.

About this task

From the panel that lists bus destinations you can use the buttons provided to create, delete, or change the mediation of destinations, or can select a destination to browse details and options for that destination.

To list bus destinations on a bus, use the administrative console to complete the following steps.

Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**. This displays a list of the bus destinations on the bus.
Any temporary destination in the list is identified by the prefix `_Q` for temporary queues or `_T` for temporary topics.

What to do next

To browse or change the properties of a destination, click its name in the list displayed.

To act on one or more of the destination listed, select the check boxes next to the names of the destinations that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

Creating a bus destination

Use the following tasks to create a new bus destination on a service integration bus.

About this task

The steps involved in defining a new destination depend on the intended usage of the destination.

These are permanent destinations, with their properties defined by administrative resources, and are in addition to any temporary destinations created at runtime by the bus for applications.
- Defining a bus destination for JMS queues.
- Defining a bus destination for JMS topics.

- Defining an alias bus destination.
- Defining a new exception bus destination

Creating a queue for point-to-point messaging:

Use this task to create a new queue, which is a bus destination configured for point-to-point messaging.

Before you begin

During this task you must specify the name of a bus member to which the bus destination is assigned. That bus member is to host the queue point for the new bus destination, and must already have been configured.

About this task

To define a new queue for point-to-point messaging use the administrative console to complete the following steps. To define a queue, you need to specify only a minimum set of properties; you can change these and other properties after you have completed this task.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that is to provide the message point for the queue.
3. In the content pane, under Destination resources, click **Destinations**. This displays a list of any existing destinations in the content pane.
4. To create a new destination, click **New** in the content pane.
   a. On the Create New Destination page, leave the type of destination as Queue.
   b. Click **Next**.
   c. In the **Identifier** field, type the name that you want to give the queue destination for administrative purposes. Restrict the name to 48 characters or less, and restrict its character set to: numerics (0–9), period (.), forward slash (/), underscore (_), percent sign (%), uppercase A–Z, lowercase a–z (but there are restrictions on the use of lowercase letters for z/OS console support). On systems using EBCDIC Katakana you cannot use lowercase characters.
5. Optional: Specify the following properties for the destination:

   Description
   An optional description of the destination, for administrative purposes.

   Maximum reliability
   The maximum reliability of messages accepted by this destination.

   Best effort nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

   For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

   Express nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

6. Click **Next**.
7. On the Select Assigned Bus Member page, select the bus member that is to provide the queue point for the destination. The queue point is where the messages for the queue are held.

   If the bus member is a WebSphere MQ server, perform the next step, otherwise omit it.
8. If the bus member is a WebSphere MQ server, set the WebSphere MQ queue point attributes:
   a. Specify a value in the **WebSphere MQ queue name filter** field, then click **Go**.

      The wizard automatically discovers available WebSphere MQ queues. However, some WebSphere MQ topologies have many thousands of queues defined to a queue manager. Use this filter to limit the number of queues that are listed.

      The default filter value is an asterisk (*). If this value (or no value) is set then all queues, or all queues of a specific type (based on any queue type custom property that is set), are listed. Any other value that you specify must meet the following criteria:

      - It must contain between 1 and 48 characters.
      - It must conform to the WebSphere MQ queue naming rules (see the WebSphere MQ topic Rules for naming WebSphere MQ objects).

      You can also use the wildcard character (*) in conjunction with other text. For example, if you enter a value of `PAYROLL*`, then all available queues with names that start with PAYROLL are displayed.

   b. Specify a WebSphere MQ queue name.

      Select a queue name from the filtered list. If the list does not include the queue that you want, select the last entry in the list labeled “other, please specify”. A text entry box is displayed next to the list box. Type the queue name into the text entry box.

      If the queue is found on the remote WebSphere MQ system, the properties of the queue as defined within WebSphere MQ are displayed as read-only fields. This should help you to confirm that you have found the queue that you want, and that it is configured as you intend. If the queue is not found, these read-only fields are removed from view.

   c. Specify the reliability levels that you require when inbound nonpersistent and inbound persistent WebSphere MQ messages are converted to service integration format messages. Applications receive messages direct from the specified WebSphere MQ queue, so in general the reliability level for a message is of no interest to the receiver because the message has already been delivered successfully. However, the message is converted to a service integration format message (and typically to a JMS format service integration message) as it is received, and this option specifies the reliability level for the service integration format message. For information about the available reliability levels, see WebSphere MQ queue points [Settings].

   d. Specify whether you want WebSphere MQ to include an MQRFH2 message header when sending messages to the queue.

      The MQRFH2 header stores service integration messaging information that does not have a corresponding WebSphere MQ message header field. When a message is sent to the destination, service integration instructs WebSphere MQ to write the message to the queue. This option specifies whether or not service integration instructs WebSphere MQ to write the message with an MQRFH2 header.

      If the consumer of the message is a JMS application running in WebSphere MQ or service integration, or a WebSphere MQ XMS application, or a WebSphere MQ MQI application that expects an MQRFH2 header, select this option. If the consumer is a WebSphere MQ MQI application that does not expect an MQRFH2 header, do not select this option.

9. Click **Next**.
10. On the Confirm Queue Creation page, review the summary of actions.
    - To create the queue, click **Finish**.
    - If you want to change any of the properties that you have specified, click **Previous**, then change the properties on the preceding pages.
11. Save your changes to the master configuration.

What to do next

If you want to change properties of the queue, see “Configuring bus destination properties” on page 1230.

By default, messages that cannot be delivered to the queue are sent to the default exception destination for the messaging engine that hosts the queue point. If you want to use a non-default exception destination for messages that cannot be delivered to this queue destination, you must have already defined that exception destination. For more information about configuring exception destinations, see Configuring an exception bus destination.

If the queue is to be used for JMS point-to-point messaging, specify the queue identifier on a JMS queue as described in “Configuring a JMS queue for the default messaging provider” on page 1590.

Creating a topic space for publish/subscribe messaging:

Use this task to create a new topic space for publish/subscribe messaging.

About this task

You need to create topic space when deploying JMS applications that use publish/subscribe messaging.

To create a topic space, you need to specify only a minimum set of required properties; you can change these and other properties after you have completed this task.

To create a topic space, use the administrative console to complete the following steps.
1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus that is to provide the publication points for the topic space.
3. In the content pane, under **Destination resources**, click **Destinations**. This displays any existing destinations in the content pane.
4. To create a topic space, click **New** in the content pane.
   a. On the Create New Destination page, select the **Topic space** option.
   b. Click **Next**.
   c. In the **Identifier** field, type the name that you want to give the topic space for administrative purposes.
5. Optional: Specify the following properties for the topic space:

   Description
   An optional description of the topic space, for administrative purposes.

   Maximum reliability
   The maximum reliability of messages accepted by this destination.

   Best effort nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

Express nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

6. Click **Next**.
7. On the Confirm Topic space creation page, review the summary of actions.
   - To create the topic space, click **Finish**.
   - If you want to change any of the properties that you have specified, click **Previous**, then change the properties on the preceding pages.
8. Save your changes to the master configuration.

What to do next

If you want to change properties of the topic space, see “Configuring bus destination properties” on page 1230.

By default, messages that cannot be delivered to the topic space are sent the default exception destination for the messaging engine that is publishing the message. If you want to use a non-default exception destination for messages that cannot be delivered to this topic space, you must have already defined that exception destination. For more information about configuring exception destinations, see Configuring an exception bus destination.

If the topic space is to be used for JMS publish/subscribe messaging, specify the topic space identifier on a JMS topic as described in “Configuring a JMS topic for the default messaging provider” on page 1591.

Creating an alias bus destination:

Use this task to create a new alias destination on a service integration bus. The alias destination can be used to route messages to a target destination in the same bus, or another (foreign) bus (including across a WebSphere MQ link to a WebSphere MQ queue). You can set its own destination properties which will override the destination defaults.

About this task

To define a new alias destination, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus on which the alias destination is to be created.

3. In the content pane, under **Destination resources**, click **Destinations**. This displays any existing destinations in the content pane.
4. To create a new destination, click **New** in the content pane.
   a. On the Create New Destination page, select Alias Destination.
   b. Click **Next**.
   c. In the **Identifier** field, type the name that you want to give the alias destination for administrative purposes.
5. Specify the following properties for the destination:

   Bus The name of the bus used by applications when referring to the alias destination. If you leave this field empty, the alias destination is created on the local bus.

   Target identifier
   The identifier of the target destination that this alias destination should route messages to.

   If the alias destination is targeting a queue provided by WebSphere MQ, type the value as a concatenation of the queue name and the queue manager name, *queue_name@qmanager_name*; for example: `Queue1@Qmgr2`.

   Target bus
   The name of the service integration bus (or foreign bus) that hosts the target destination. This can be the name of a foreign bus that represents a WebSphere MQ network. The default is the name specified for the **Bus** property.
6. Optional: Specify the following properties for the destination. These will override the destination defaults.

   Description
   An optional description of the destination, for administrative purposes.

   Enable producers to override default reliability
   Controls the quality of service for message flows between producers and the destination. Select this option to use the quality of service specified by producers instead of the quality defined for the destination.

   Default reliability
   The reliability assigned to a message produced to this destination when an explicit reliability has not been set by the producer.

   INHERIT
   Use the reliability configured on the target destination.

   Best effort nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

   For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Default reliability property of the bus destination.

   Express nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

Maximum reliability
The maximum reliability of messages accepted by this destination.

Producers cannot send messages to this destination with a reliability higher than the value specified for this property.

INHERIT
Use the reliability configured on the target destination.

Best effort nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

Express nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

Send allowed
Clear this option (setting it to false) to stop producers from being able to send messages to this destination..

INHERIT
Use the value configured on the target destination.

TRUE Producers can send messages to this destination.

FALSE
Producers cannot send messages to this destination.

Receive allowed
Clear this option (setting it to false) to prevent consumers from being able to receive messages from this destination.

INHERIT
Use the value configured on the target destination.

TRUE Consumers can receive messages from this destination.

FALSE
Consumers cannot receive messages from this destination.

Default priority
The default priority for messages sent to the target destination, in the range 0 (lowest) through 9 (highest), or -1. The value -1 indicates that messages should use the default priority defined on the target destination. The default priority is used only if a message does not specify its own priority.

7. Click **Next**.
8. On the Confirm Destination Creation page, review the summary of actions.
   - To create the alias destination , click **Finish**.
   - If you want to change any of the destination properties, click **Previous**, then change the properties on the preceding pages.
9. Save your changes to the master configuration.

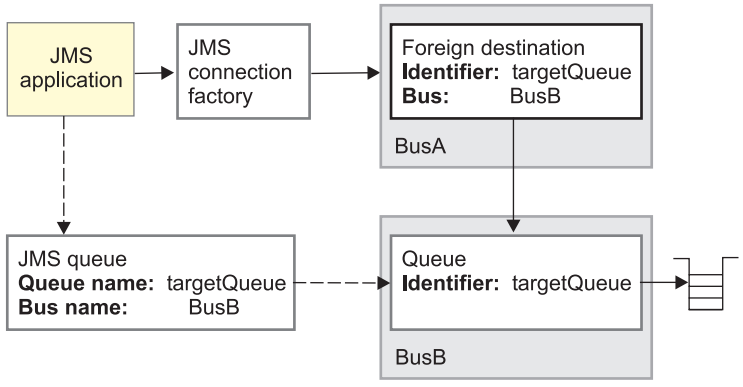
What to do next

If you want to change properties of the alias destination, see “Configuring bus destination properties” on page 1230. Further advanced configuration to the alias can be made using the “Configuring alias bus destination properties” on page 1234 panel, if required.

Creating a foreign bus destination:

Use this task to create a new foreign destination on a service integration bus. A foreign destination provides a mapping to a destination of the same name on a different bus, and enables applications on one bus to access directly the destination on another bus. You can set its own destination properties which will override the destination defaults.

About this task



The foreign destination encapsulates the name of the target destination that exists in the foreign bus (Identifier property) and the name of that foreign bus (Bus property). An application that wants to use the foreign destination to exchange messages with the target destination must specify the Identifier and Bus properties.

For example, an administrator wants JMS applications to connect to one bus, BusA, and send messages to a JMS queue backed by a queue, targetQueue, on another bus, BusB. The administrator connects the buses, creates a foreign destination on BusA and sets the following properties on the foreign destination and JMS queue:

JMS queue	Foreign destination (on BusA)	Queue (on BusB)
Queue name targetQueue **Bus name** BusB	**Identifier** targetQueue **Bus** BusB	**Identifier** targetQueue

To define a new foreign destination, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the bus on which the foreign destination is to be created.
3. In the content pane, under **Destination resources**, click **Destinations**. This displays any existing destinations in the content pane.
4. To create a new destination, click **New** in the content pane.
   a. On the Create New Destination page, select Foreign Destination.
   b. Click **Next**.
   c. In the **Identifier** field, type the name of the target destination that exists in the foreign bus. The Identifier must match the name of the target destination that exists in the foreign bus.
   d. In the **Bus** field, type the name of the foreign bus that hosts the target destination. This must be the name of a foreign bus administrative object that is already defined on the bus on which the foreign destination is to be created.
5. Optional: Specify the following properties for the destination. These will override the destination defaults.

   Description
   An optional description of the destination, for administrative purposes.

   Default reliability
   The reliability assigned to a message produced to this destination when an explicit reliability has not been set by the producer.

   Best effort nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

   For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

   Express nonpersistent
   Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

   Reliable nonpersistent
   Messages are discarded when a messaging engine stops or fails.

   Reliable persistent
   Messages may be discarded when a messaging engine fails.

   Assured persistent
   Messages are not discarded.

Maximum reliability
The maximum reliability of messages accepted by this destination.

Best effort nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

Express nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

Default priority
The default priority for messages sent to the target destination, in the range 0 (lowest) through 9 (highest), or -1. The value -1 indicates that messages should use the default priority defined on the target destination. The default priority is used only if a message does not specify its own priority.

6. Click **Next**.
7. On the Confirm Destination Creation page, review the summary of actions.
   - To create the foreign destination , click **Finish**.
   - If you want to change any of the destination properties, click **Previous**, then change the properties on the preceding pages.
8. Save your changes to the master configuration.

What to do next

Ensure that you have defined a foreign bus (to identify the target bus) and the target destination on that bus.

Configuring bus destination properties

Use this task to browse or change the configuration properties of a bus destination.

About this task

To browse or change the properties of a destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.

2. Change the destination properties to suit your needs. See Learning about bus destinations for detailed description of bus destinations.
3. Click **OK**.
4. Save your changes to the master configuration.

Specifying an exception bus destination:

Specify an exception destination if you want to prevent messages that cannot be delivered to their original intended destination from being lost.

About this task

Complete this task if you want a bus destination to use a non-default exception destination for undeliverable messages. The bus destination for which you are specifying an exception destination is referred to as the “intended bus destination”.

Note:

- It is not possible to configure an exception destination for the entire bus; you must configure an exception destination for each destination on the bus.
- You should not modify or delete the default system destination.
- Configuring an exception destination prevents guaranteed ordering throughout the bus. For more information on ensuring ordering, see Message ordering.

For more information on exception destinations see **Related links**.

To specify a new exception destination for a bus destination, use the administrative console to complete the following steps:

1. Create the exception destination, as a queue. For information about creating a queue, see “Creating a queue for point-to-point messaging” on page 1222.
2. Type the name of the exception destination in the **Exception destination** field of the intended bus destination.
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
   b. Under **Exception destination**, select the **Specify** radio button.
   c. Type the name of the exception destination in the **Exception destination** field.
3. Save your changes to the master configuration.

Controlling whether applications can send or receive messages for a bus destination:

Use this task to change the configuration properties of a bus destination to control whether applications can send messages to, or receive messages from, the destination.

About this task

For example, some destinations only exist in order to be associated with mediations; applications should not be able to put to or get from such a destination.

The changes made in this task affect the configuration of a bus destinations and when saved, are automatically applied to all message points for that destination. You can make the same changes to an individual destination localization point to control access to only that one point.

When you save changes that affect the access to a bus destination, this affects producers or consumers attached to message points for that destination. For more information about the effect on producers or consumers, see The effect on producers or consumers of changing access to a bus destination.

To prevent applications from either sending messages to, or receiving messages from, a destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
2. Optional: Change one or more of the following properties:

   Receive allowed

   Clear this check box (setting to the option to false) to prevent messages from being received from message points for this destination. The effect depends on the type of destination:

   - Queue point. Any open consumers change state and an exception is thrown when the consumer requests a message.
   - Publication point. Any messages that have been published to the messaging engine for a publication point are stopped from proceeding either to local consumers or onwards to other messaging engines. Local consumers get the same exception as for a queue point.
   - Mediation point of a mediated destination. The bus stops the mediation instance that is running locally to the mediation point; other instances of the mediation running on other messaging engines continue as normal.

   In all cases, messages can continue to be sent, and accumulate on the destination localization point.

   Send allowed

   Clear this check box (setting the option to false) to prevent messages from being accepted onto the message points for this destination.

   - For a queue point of a non-mediated destination, or a mediation point of a mediated destination, new messages (from attached producers or forwarded from another destination) are redirected to any available message point. If no message points are available, then messages that have already been accepted onto the bus, and new messages from attached producers, are preserved by the bus until a message point becomes available. The only exception to this is the case of a destination with only one message point (queue point or mediation point depending on whether the destination is mediated or non-mediated), where the producer is attached to the same messaging engine. In this case, an exception is thrown on each send call. The exception message indicates that the reason for the exception is that the only extant localization has been disabled for send. The producer remains open as normal, and any more send calls succeed if the **Send allowed** property of the localization is reselected (reset to true).
   - For a queue point of a mediated destination, clearing this **Send allowed** property alters the behavior of the mediation instances that are sending to the destination in the same way as setting it to false on a non-mediated destination affects producing applications: Messages are sent instead to any alternative message point. If no localizations are available, are preserved by the bus until a message point becomes available. For any mediation instance (that is, on any server that has a mediation point), if the same server hosts a queue point, and that queue point is the only queue point for the destination, then the mediation changes to the “stopped on error” state.
   - For a publication point, clearing this **Send allowed** property stops applications attached locally to the topic space from publishing messages. The send calls receive an exception, and the producer remains open.

   Receive exclusive

   If you select this check box (setting to the option to true), then only one consumer can be attached to any message point. This property is particularly intended for use with queues, but can be used with any type of destination.

   - For a queue, the bus chooses a queue point for each request to create a consumer. If the selected queue point already has an attached consumer, then the call fails with an exception

(containing an exception message and linked exception that indicate the precise nature of the failure). There is no guarantee that all available queue points are used before the exception is thrown.

- For a topic space, only one consumer can attach to any given messaging engine.

If you change the **Receive exclusive** property from false to true, some consumers are selected to be the exclusive receivers in accordance with the rules above. All other consumers are detached from the destination, in the same way as is described above for a transition of the **Receive allowed** property from true to false.

3. Click **OK**.
4. Save your changes to the master configuration.

Specifying whether strict message order is preserved for a bus destination:

Use this task to change the configuration properties of a bus destination to control whether strict message order is preserved.

Before you begin

The changes made in this task affect the configuration of a bus destination. Before you begin this task, make sure that you fully understand the restrictions that apply to ordered destinations. For further details see Strict message ordering for bus destinations.

About this task

Although messages produced by a single producer to a single destination are seen by a consumer to that destination in the same order as they are produced, there are certain topologies and events, such as system failures, that can change the order of messages (see Message ordering). If strict message order is essential, you can configure a destination to preserve the order of the messages in a much more rigorous manner. Configuring a destination in this way, when used in conjunction with restricted topologies, maintains message order in all circumstances (for further information, see Strict message ordering for bus destinations).

To preserve the order of the messages that are delivered to a destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
2. Select the **Maintain strict message order** check box (setting the option to true).
3. Click **OK**.
4. Save your changes to the master configuration.

Results

An ordered destination has priority over several other configuration properties, such as the number of consumers that are allowed to attach to the destination. The system overrides these configuration properties at run time, and generates a system log warning. For more information about the configuration properties that the system overrides at run time, and the impact of including an ordered destination in your system, see Strict message ordering for bus destinations.

Specifying whether messages are forwarded to WebSphere MQ as JMS messages:

Use this task to create a destination context property that determines whether an MQRFH2 header is added to WebSphere MQ JMS messages produced by foreign or alias destinations.

Before you begin

This task assumes that you have created an alias bus destination or a foreign bus destination. For more information, see “Creating an alias bus destination” on page 1225 or “Creating a foreign bus destination” on page 1228.

About this task

You only need to carry out this task if you want JMS messages produced by foreign bus destinations or alias bus destinations forwarded to WebSphere MQ. In this task, you define a destination context property called _MQRFH2Allowed that adds an MQRFH2 header to JMS messages. If you do not configure _MQRFH2Allowed, the default value is NO, and an MQRFH2 header is not added to the message. WebSphere Application Server Version 5.1 users may be aware of a WebSphere MQ flag called TARGCLIENT that was used to add RFH2 headers to JMS messages. For more information about the use of the MQRFH2 header in interoperating with WebSphere MQ, see How service integration converts the message body to and from WebSphere MQ format and Point-to-point messaging with a WebSphere MQ network.

To create and configure a context property called _MQRFH2Allowed on a foreign or alias destination, use the administrative console to complete the following steps:

1. Display the context properties for a selected foreign or alias destination:
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**.
   b. Select the foreign or alias destination for which you want to create the context property.
   c. Under **Additional Properties**, click **Context properties**.
2. Click **New** to create a new context property.
3. Specify the following context information:

   Name Type the name `_MQRFH2Allowed`.

   Context type
   Select the information type `Boolean` in the selection list.

   Context value
   Type the value `true` .
4. Click **OK**.
5. Save your changes to the master configuration.

Results

An MQRFH2 header is added to all messages produced by the foreign or alias destination. This means that JMS messages will be forwarded to WebSphere MQ as JMS messages.

Configuring alias bus destination properties:

Use this task to browse or change the configuration properties of an alias bus destination. The alias destination can be used to route messages to a target destination in the same bus, or another (foreign) bus (including across a WebSphere MQ link to a WebSphere MQ queue). Any alias bus destination properties that you set will override the properties of the target destination, otherwise the target destination’s properties will apply.

About this task

To browse or change the properties of an alias destination, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses**.

2. In the content pane, click the name of the bus on which the alias destination was created.
3. In the content pane, under **Destination resources**, click **Destinations**. This displays any existing destinations in the content pane.
4. Select the alias destination that you want to configure.
5. You can change the following alias destination properties to suit your needs. These values will override the destination defaults.

Description
An optional description for the bus destination, for administrative purposes.

Target identifier
The name of the destination for which this is an alias.

Target bus
The name of the bus on which the destination for which this is an alias exists.

Use all target queue points
This property box is displayed only if the destination identified by the Target identifier and Target bus properties has multiple queue points configured (that is, deployed to a cluster bus member with multiple messaging engines) and the Target bus is the local bus.Whether to use all target queue points indicates whether all the queue points of the target queue can be used, including any queue points created after the alias is configured. When selected, the **Target queue points** menu is disabled.

Unselected queue points
A list of the queue points that are not addressable by the alias definition. The list is generated from the complete list of queue points for this queue. This list is enabled only when **Use all target queue points** is unchecked.

Selected queue points
A list of the queue points that are addressable by the alias definition. This list is enabled only when **Use all target queue points** is unchecked

Enable producers to override default reliability
Controls the quality of service for message flows between producers and the destination. Select this option to use the quality of service specified by producers instead of the quality defined for the destination.

Default reliability
The reliability assigned to a message produced to this destination when an explicit reliability has not been set by the producer.

INHERIT
Use the reliability configured on the target destination.

Best effort nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

Express nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

Maximum reliability
The maximum reliability of messages accepted by this destination.

Producers cannot send messages to this destination with a reliability higher than the value specified for this property.

INHERIT
Use the reliability configured on the target destination.

Best effort nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable and as a result of constrained system resources.

For non-transactional JMS message-driven beans and MessageListeners that use a JMS destination configured on the default messaging provider, best-effort nonpersistent messages are not recoverable. In this case, if a message is unlocked because the message-driven bean or MessageListener threw an exception, then the message is not redelivered or sent to the exception destination because it was deleted from the message store when it was passed to the listener. If you require higher message reliability for non-transactional JMS message-driven beans and MessageListeners, configure a different option for the Maximum reliability property of the bus destination.

Express nonpersistent
Messages are discarded when a messaging engine stops or fails. Messages may also be discarded if a connection used to send them becomes unavailable.

Reliable nonpersistent
Messages are discarded when a messaging engine stops or fails.

Reliable persistent
Messages may be discarded when a messaging engine fails.

Assured persistent
Messages are not discarded.

Default priority
The default priority for messages sent to the target destination, in the range 0 (lowest) through 9 (highest), or -1. The value -1 indicates that messages should use the default priority defined on the target destination. The default priority is used only if a message does not specify its own priority.

Send allowed
Clear this option (setting it to false) to stop producers from being able to send messages to this destination..

INHERIT
Use the value configured on the target destination.

TRUE Producers can send messages to this destination.

FALSE
Producers cannot send messages to this destination.

Receive allowed
Clear this option (setting it to false) to prevent consumers from being able to receive messages from this destination.

INHERIT
Use the value configured on the target destination.

TRUE Consumers can receive messages from this destination.

FALSE
Consumers cannot receive messages from this destination.

Reply destination
The name of a destination to be appended to any non-empty reverse routing path of messages sent to this destination.

Reply destination bus
The bus on which the reply destination exists.

Default forward routing path
The value to which a message's forward routing path will be set if the message contains no forward routing path. This identifies a sequential list of intermediary bus destinations that messages must pass through to reach a target bus destination. The format of the field is a list of line-delimited bus destinations specified as *bus*:*name*.

Delegate authorization check to target destination
Indicates whether the destination access role checked when a user accesses the alias destination is that of the target destination (checked) or the alias destination (unchecked). ″

WebSphere MQ RFH2 allowed
WebSphere MQ RFH2 is allowed.

6. Click **OK**.
7. Save your changes to the master configuration.

What to do next

If you want to enable correct processing of messages, under Additional Properties click **Context properties**. This information adds to the context information derived from processing the message header. If you want an expandable tree view of all the applications and messaging resources that reference the current destination, both directly and indirectly, under Related Items click **Application resources topology**. As many of the references as possible are resolved to links to the associated configuration panel for the referenced object.

Configuring mediations

Use this task to configure one or more mediations for a selected bus destination in a service integration bus.

About this task

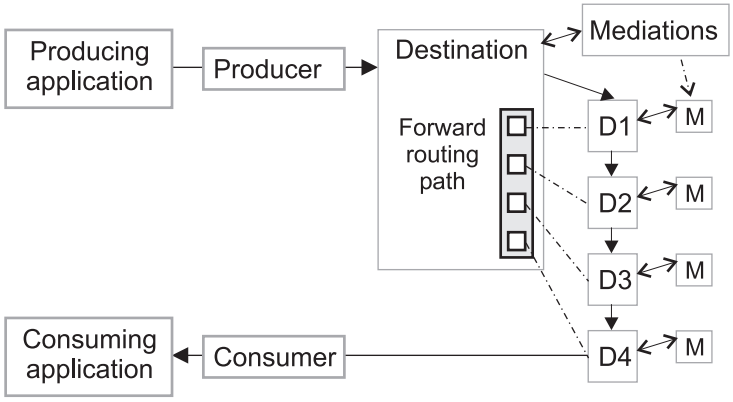
You can configure a destination with one or more mediations that refine how messages are handled by the destination, and can configure destination properties that affect how the mediations are used; for example, that they should run within a global transaction.

For more information about configuring mediations, see "Mediating a destination" on page 1264.

Configuring a destination forward routing path

Use this task to configure a forward routing path for a selected bus destination. The forward routing path identifies a sequence of bus destinations that a message should pass through after it has been delivered to the bus destination to which the producer is attached.

About this task



If producing and consuming applications both attach directly to the destination that is to host and process their messages, you do not need to use routing paths. You can still use mediations assigned to the destination to manipulate messages. However, if you want a more flexible architecture, you can assign mediations to several destinations then specify those destinations as the forward routing path for another destination. For example in the above diagram, on the destination called Destination, on the bus called myBus, you would set the Default forward routing path property to:

```
mybus:D1
mybus:D2
mybus:D3
mybus:D4
```

For each destination that should be visited by a reply message on its ways back to the producer application, you can specify the name of a *reply destination*, which is the next destination in the reverse routing path. If the same destinations are used for both the forward and return routing paths, you can configure the return routing path when you configure the forward routing path, as described in this topic. Otherwise, see “Configuring a destination reverse routing path” on page 1239.

To configure a forward routing path, you only need to change the properties of the destinations you want to use. This topic contains optional steps to create destinations, in case you have not already done so.

To configure a forward routing path for a destination, use the administrative console to complete the following steps:

1. Optional: If you have not already done so, create the destination to which the producer applications attach. For this destination, you can create a queue or create a topic space.
2. Optional: If you have not already done so, create each destination in the path. Only the last destination in the path can be a topic space; all other destinations in the path must be queues.

   If the destination is also to be part of the return routing path for reply messages, specify the name of the next destination in the return routing path:

   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
   b. In the Reply destination bus field, type the name of the next destination in the return routing path.
   c. Save your changes to the master configuration.

3. Optional: If you have not already done so, assign mediations to the destinations in the path. You can include a destination without mediations in a routing path, to provide a future option to apply a mediation assigned to that destination.
4. Edit the properties of the destination to which the producer applications attach, to configure the forward routing path.
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
   b. In the Default forward routing path field, type the names of the destinations in the path. Type a list of bus destinations, each on a separate line, of the form *bus_name*:*destination_name*

      Where

      bus_name
      Is the name of the service integration bus on which the destination is configured.

      destination_name
      is the name of the bus destination.

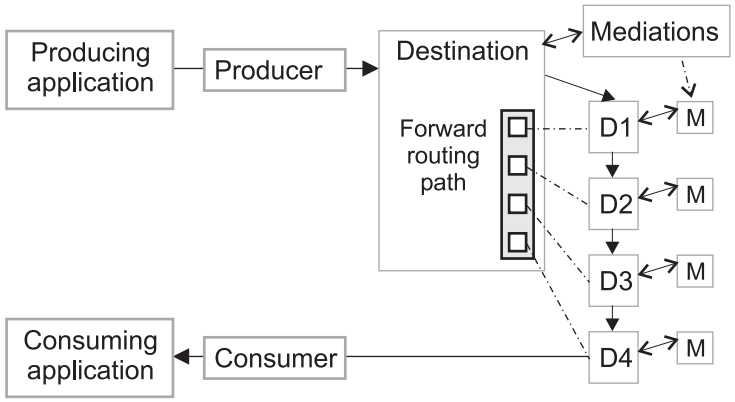
      For example:

      ```
      mybusA:queueA
      anobusB:queueB
      busC:queueC
      ```
   c. Save your changes to the master configuration.

Configuring a destination reverse routing path

Use this task to configure a reverse routing path between two bus destinations. The reverse routing path identifies the list of destinations that any reply message should sent to from the consumer back to the producer.

About this task



If producing and consuming applications both attach directly to the destination that is to host and process their messages, you do not need to use routing paths. You can still use mediations assigned to the destination to manipulate messages. However, if you want a more flexible architecture, you can assign mediations to several destinations then specify those destinations as the forward routing path for another destination.

For each destination that is to be part of a reverse routing path back to the producer application, you can specify the name of a *reply destination*, which is the next destination in the reverse routing path. For example in the above diagram, if reply messages are to retrace the forward routing path back to the producing application, you would set reverse routing path as follows:

Destination	Reply destination bus
D4	D3
D3	D2
D2	D1
D1	Destination

You can specify a different reverse routing path to the forward routing path, to enable more mediations to be applied to reply messages, or to apply mediations in a different destination sequence.

If the same destinations are used for both the forward and reverse routing paths, you can configure the reverse routing path when you configure the forward routing path, as described in “Configuring a destination forward routing path” on page 1238. Otherwise, you can configure (or change) the reverse routing path as described in this topic.

To configure a reverse routing path, you only need to change the properties of the destinations you want to use. This topic contains optional steps to create destinations, in case you have not already done so.

To configure a reverse routing path for a destination, use the administrative console to complete the following steps:

1. Optional: If you have not yet done so, create the destination to which the producer applications attaches. For this destination, you can create a queue or create a topic space.
2. Optional: If you have not already done so, create each destination in the path.
3. Optional: If you have not already done so, assign mediations to the destinations in the path. You can include a destination without mediations in a routing path, to provide a future option to apply a mediation assigned to that destination.
4. On each destination in the reverse routing path, specify the name of the next destination in the path:
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
   b. In the Reply destination field, type the name of the next destination in the reverse routing path.
   c. In the Reply destination bus field, type the name of the next destination in the reverse routing path.
   d. Save your changes to the master configuration.

Configuring context properties for a bus destination

Use this task to configure context properties for a bus destination.

About this task

The context properties contribute to the mediation context that is used in conjunction with the message header information to ensure that messages are processed correctly by a mediation assigned to this bus destination.

Note: Context properties on the destination take precedence over context properties on a mediation. For example, if a property with the same name is configured for a destination and a mediation, the property on the destination takes precedence.

To configure context information for a bus destination, use the administrative console to complete the following steps:

1. Display the context properties for a selected bus destination:
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
   b. Under **Additional Properties**, click **Context properties**.

2. Choose whether to create a new context property, configure an existing property, or delete one or more existing properties.
   - To create a new context property, click **New**.
   - To browse or change an existing context property, click the property name.
   - To delete one or more existing context properties, select the check box for each property you want to delete, then click **Delete**.
3. To create or change a context property, specify the following context information:

   Name Type a name for the context property.

   Context type
   Select the information type for the context property in the selection list.

   Context value
   Type a value for the context property.
4. Click **OK**.
5. Save your changes to the master configuration.

Administering destination roles

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups must have authority to perform messaging operations, at a bus destination. By administering destination roles, you can control which users and groups can perform operations at a bus destination, and the type of operation that they can perform.

About this task

You use the administrative console to administer users and groups in access roles for a destination. The access roles available for a destination depend on the type of destination. The table below lists the roles that you can assign for each destination type:

Destination type	Access roles
queue	sender, receiver, browser, creator
port	sender, receiver, browser, creator
webService	sender, receiver, browser, creator
topicSpace	sender, receiver
foreignDestination	sender
alias	sender, receiver, browser

In addition to controlling which users and groups have access to a specific local or foreign destination, you can also control the inheritance of access roles for a specific local destination. In this case, the default access roles that apply to all the destinations in the local bus namespace are added to any access roles that have been added for a specific destination.

Use the following tasks to administer destination roles.
- Listing users and groups in bus destination roles by using the wsadmin tool
- Adding users and groups to bus destination roles by using the wsadmin tool
- Removing users and groups from bus destination roles by using the wsadmin tool
- Defining destination defaults inheritance by using the wsadmin tool
- Determining destination defaults inheritance by using the wsadmin tool

Adding users and groups to destination roles:

Service integration bus security uses role-based authorization. By adding users and groups to the destination roles for a secured bus, you can control which users and group members can perform messaging operations at a bus destination.

Before you begin

Ensure that the following conditions are met:

- Security is enabled for the bus. For more information, refer to “Securing buses” on page 1272.
- The users and groups that you want to add to destination roles must exist already in the user repository.

About this task

By adding users or groups to destination role, you grants the users or groups authority to perform the operation defined by the role at a selected destination. The destination roles are sender, receiver, browser, and creator, depending on the destination type.

In this task you use the administrative console Security wizard to retrieve selected users or groups from the user repository, and add them to destination roles for selected bus destinations.

Note: To add a large number of users to destination roles, it is advisable to create a group in the user repository, and add the group to the destination roles.

1. Start the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destination to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles for this destination. You can verify that the user or group you want to add does not already have a role at this destination.
5. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to access roles for the expanded destination:
   a. Search for the users or groups that you want to add to access roles for the expanded destination:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to access roles for the currently expanded destination, and click **Next**. A list of user IDs or group names that you can add to destination roles is displayed. Note that some users or groups may already be assigned to access roles for this destination.
   d. Select the appropriate access role icon for the user ID or group name that you want to add to the role at this destination. For example, select the **Receiver** icon for a user ID or group name that you

want to add to the receiver role. The icon changes from  to  to show that you have added the user or group to the access role for the resource.

e. Repeat the previous step to add more users or groups to access roles for the currently expanded destination, and then click **Next**. A summary of your access role assignments is displayed.

f. Click **Previous** to review and change your assignments, if required.

g. Click **Finish** to confirm your assignments.

6. Repeat steps 4 and 5 for each destination you want to work with.
7. Save your changes to the master configuration.

Results

The selected users and groups are added to the access roles for the currently expanded destination. The new access role assignments are displayed in the Destination access roles panel.

Example

A group called MyGroup receives messages from three queues, Queue 1, Queue 2, and Queue 3. If you want the group MyGroup to produce and consume messages at an additional destination, Queue 4, you add MyGroup to Queue 4, and then add MyGroup to the sender and receiver roles for Queue 4.

What to do next

Use the administrative console to perform other security administrative tasks.

Removing users and groups from destination roles:

Service integration bus security uses role-based authorization. By removing users and groups from the destination roles for a secured bus, you can prevent those users and group members from performing messaging operations on the bus.

About this task

When selected users and groups no longer require access to a destination, you can remove them from all the roles for that destination.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles** A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destinations to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles at this destination, and verify that the user or group that you want to remove has a role at this destination.
5. Select the users and groups that you want to remove from all role types at this destination, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from all role types at the selected destination. The Manage access roles for users and groups panel displays the updated role type assignments.

Example

The members of three groups, Group A, Group B, and Group C, belong to the sender role and the receiver role for two queue destination, Queue 1 and Queue 2. If Group B is no longer required to send and receive messages on Queue 2, you can use this task to remove Group B from all the role types on Queue 2.

What to do next

You can perform other security administration tasks by using the administrative console.

Listing users and groups in destination roles:

Service integration bus security uses role-based authorization. By listing the users and groups in the destination roles for a selected secured bus, you can find out which users and groups are authorized to access the bus, and its resources.

About this task

In this task you use the administrative console to list all the users and groups in destination roles for selected destinations. The list includes users and groups that have references in the service integration role-based configuration; it does not include all the users and groups that exist in the user repository. The permitted destination roles are sender, receiver, browser and creator, depending on the destination type. Icons are used in the administrative console to represent the roles to which users and groups have been assigned. For example, if the role type set icon () is displayed in the sender role for a group called Group 1, it means that Group 1 has been assigned to the sender role for a selected destination. For a complete description of all the icons used to represent role assignments in the administrative console, refer to Access role assignments for bus security resources.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destinations panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination header.

Results

The Destination access roles panel lists the users and groups in access roles for the expanded destination.

What to do next

You can now administer the users and groups in destination roles at this destination.

Restoring default inheritance for a destination:

Service integration bus security uses role-based authorization. By default, all local destinations inherit access roles from the default resource. If default inheritance has been previously overridden, you can restore it for a selected destination.

Before you begin

Default inheritance has been overridden for a selected secured destination. For more information, refer to Overriding inheritance from the default resource for a destination.

About this task

If default inheritance has been overridden for a particular destination, you can override it. In this task, you use the administrative console to restore the role type assignments from the default resource to a selected destination. A destination can only inherit access roles that are allowed for that particular type of destination. For example, a topic space can inherit the sender and receiver roles, but it cannot inherit the browser role. Inherited access roles are added to any existing access roles for the destination.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destinations panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination to list the users and groups that have been assigned to roles for this destination.
5. Select the **Inherit from default** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The role type assignments for the default resource are inherited by the selected destination. The Destination access roles panel displays the newly inherited default access roles for the destination, and any existing access roles.

What to do next

You can perform other security administration tasks by using the administrative console.

Disabling inheritance from the default resource:

This task enables you to disable the inheritance of security access roles from the default resource for selected resources.

Before you begin

About this task

In this task, you use the administrative console to disable the inheritance of access roles from the default resource for selected resources. Disabling inheritance from the default access roles means that users or groups that have roles in the default access role no longer have access to this destination.

Use the administrative console to disable the inheritance of access roles from selected resources as follows:

1. Use the administrative console to navigate to a list of the resources you want to work with for a specific bus. For example, to list all the known destinations for a selected bus, click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destination panel lists all the destinations defined on the selected bus.
2. Do one of the following to select one or more resources:
   - Click the name of a single resource.
   - Select the check boxes next to multiple resources, and click **Manage Access Roles**.

   The Access Roles panel displays access role information for each selected resource within a collapsed section.
3. Expand a resource header to list the users and groups that have been assigned to roles for this resource.
4. Clear the **Inherit from default** checkbox.
5. Click **OK** to save your changes. The role types for the default resource are removed from the selected resource, and the Access Roles panel is updated.
6. Save your changes to the master configuration.

Deleting a bus destination

Use these tasks to delete a bus destination from a service integration bus.

About this task

The steps required depend on the type of destination.

Deleting a non-topic space bus destination:

Use this task to delete, from a service integration bus, a bus destination that is not for a topic space.

Before you begin

You cannot delete bus destinations that are required by the system; for example, the default exception destination for a messaging engine: `_SYSTEM.Exception.Destination.`*`me_name`*.

Before you can completely delete a bus destination, you must ensure that the destination message point is empty of all messages. A delete action is rejected if there are any messages on the message point, either committed, uncommitted, indoubt or locked.

Any reliable or durable messages sent after the delete action, but before the close of producers, are sent to one of the following destinations:

- Another message point of the destination, if possible.
- The exception destination for the deleted destination.

Note: Any attached producers or consumers are closed asynchronously, and do not prevent the delete action. Any messages sent after the delete action, but before close of producers might not be discarded. Depending on the options set, the messages may be moved to the exception destination. For more information see “Specifying an exception bus destination” on page 1231 and Exception destination.

About this task

To delete a destination, use the administrative console to complete the following steps.

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**.

2. Click the check box next to the destination name.
3. Check for and resolve any locked messages. Messages that are locked, as part of in-flight transactions or as in-doubt messages, are not deleted.
4. To clear all messages from the destination, click **Clear**. This action deletes all of the messages that it can access. Messages that are locked, part of in-flight transactions or in-doubt are not deleted. If there are messages that cannot be deleted, they are skipped and the operation indicates that it was partially successful.
5. To delete the empty destination, click **Delete**.
6. To confirm that you want to delete the destination click **OK**.

Results

The bus destination is deleted.

What to do next

Monitor the exception queue for the messaging engine to which the destination was assigned, to see if there are any in-transit messages that were not handled by clearing the destination before it was deleted.

Deleting a topic space:

Use this task to delete, from a service integration bus, a topic space.

Before you begin

Before you can delete a topic space, you must ensure that the publication points are empty. A delete action is rejected if there are any indoubt or uncommitted publications for a publication point.

Note: Any attached producers are closed asynchronously, and do not prevent the delete action. However, any messages sent after the delete action, but before close of producers are discarded.

The service integration bus topic space is the primary messaging object upon which WS-Notification depends at run time. For information about the effect that deleting a topic space has upon new and existing WS-Notification applications, see Failures as a result of changes in topic space and topic namespace configurations.

About this task

To delete a topic space, use the administrative console to complete the following steps.

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**.
2. Click the check box next to the destination name.
3. Check for and resolve any undelivered publications (indoubt or uncommitted) for the topic space. Messages that are locked, as part of in-flight transactions or as in-doubt messages, are not deleted.
4. To delete the empty topic space, click **Delete**.
5. To confirm that you want to delete the topic space, click **OK**.
6. Save your changes to the master configuration.

Results

The topic space is deleted.

Resetting a destination

Use this task to reset a bus destination.

About this task

You may want to reset a destination if, for example, it has become corrupt. To do this, you use a wsadmin AdminControl command.

1. Use a wsadmin AdminControl command of the following form to find the name of the messaging engine.
   - Using Jython:
     ```
     me = AdminControl.queryNames("type=SIBMessagingEngine,name=messaging_engine_name,* " )
     ```
   - Using Jacl:
     ```
     set me [$AdminControl queryNames type=SIBMessagingEngine,name=messaging_engine_name,*]
     ```

   Where *messaging_engine_name* is the name of the messaging engine.
2. Use the wsadmin AdminControl command of the following form to reset the destination.
   - Using Jython:
     ```
     AdminControl.invoke(me, "resetDestination", ["destination"] )
     ```
   - Using Jacl:
     ```
     $AdminControl invoke $me resetDestination {"destination"}
     ```

   Where *destination* is the name of the destination.

Results

The bus destination is reset ready for use.

Managing bus destinations with administrative commands

You can use these commands to manage bus destinations.

About this task

These commands provide an alternative to using the administrative console or using the more complex syntax of wsadmin and JACL.

1. Open a wsadmin command session in local mode For example:
   ```
   wsadmin -conntype none -lang jython
   ```

   Note: The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.
2. Type `AdminTask.`*command*

   Where *command* is the command format as indicated in the related reference topics; for example:
   ```
   wsadmin>AdminTask.listSIBDestinations(["-bus", "abus"])
   '(cells/9994GKCCell01/buses/abus|sib-destinations.xml#SIBTopicSpace_1098181446388)
   (cells/9994GKCCell01/buses/abus|sib-destinations.xml#SIBQueue_1098181503600)
   (cells/9994GKCCell01/buses/abus|sib-destinations.xml#SIBQueue_1098184221748)'

   wsadmin>AdminTask.listSIBDestinations(["-bus", "abus", "-type", "TopicSpace"])
   '(cells/9994GKCCell01/buses/abus|sib-destinations.xml#SIBTopicSpace_1098181446388)'
   ```

Configuring message points

Use the following tasks to configure message points on service integration buses.

About this task

A message point is the location (for example, queue point or publication point) at which messages for a bus destination are held in a messaging engine. The runtime state of the message point represents a runtime instance of the destination.

When you define a new bus destination, its message point is created. You can then configure some properties of the message point to override the general properties defined for the destination.

You can also administer the messages on the runtime view of the message point.
- "Configuring a message point"
- "Listing message points for a bus destination"
- "Listing message points for a messaging engine"

Listing message points for a messaging engine

Use this task to list the message points for bus destinations on a selected messaging engine.

About this task

To display a list of message points for a messaging engine, use the administrative console to complete the following steps.

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines**. This displays a list of messaging engines on the bus.
2. Click the name of the messaging engine.
3. Under Message Points, click the link for the type of point you want to list.

Results

A list of message points for the selected messaging engine is displayed in the content pane.

Listing message points for a bus destination

Use this task to list the message points for a selected bus destination.

About this task

To display a list of message points for a bus destination, use the administrative console to complete the following steps.

1. Display the bus destination.
2. In the content pane, under Message Points, click the link for the type of point you want to list.

Results

A list of message points for the selected bus destination is displayed in the content pane.

Configuring a message point

Use this task to browse or change the configuration properties of a message point for a bus destination.

About this task

You can configure some properties of the message point to override the general properties defined for the destination.

To browse or change the properties of a message point, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
2. Under Message Points, click the link for the type of message point:
   - **Queue points**, for a queue
   - **Publication points**, for a topic space
   - **Mediation points**, for a mediated queue or topic space

3. In the content pane, click the name of the message point.
4. Optional: Change the message point settings.
5. Click **OK**.
6. Save your changes to the master configuration.

Managing messages on message points

Use these tasks to list and act on runtime messages that exist on message points in a service integration bus.

About this task

You can list the message points for bus destinations and messaging engines, and list the messages on a selected message point. You can use the list of messages as part of a troubleshooting task to find messages that need to be deleted.

- "Listing messages on a message point"
- "Deleting messages on a message point" on page 1251

Listing messages on a message point

Use this task to list the messages that exist on a message point for a selected bus destination or messaging engine.

About this task

To display a list of messages on a message point, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the service integration bus.
3. Optional: To list the message points for a bus destination, complete the following steps:
   a. In the content pane, under **Destination resources**, click **Destinations**.
   b. Click the destination name.
4. Optional: To list the message points for a messaging engine, complete the following steps:
   a. In the content pane, under **Topology**, click **Messaging engines**
   b. Click the messaging engine name.
5. Under Additional Properties, click **Message points** This displays a list of message points in the content pane.
6. Click the message point name. This displays the properties of the destination localization in the content pane.
7. Click the Runtime tab.
8. Under Additional Properties, click **Messages**

Results

A list of messages on the selected message point is displayed in the content pane.

What to do next

You can select one or more messages to act on; for example, to display the message content, delete messages.

Deleting messages on a message point

Use this task to delete one or messages that exist on a message point for a selected bus destination or messaging engine.

About this task

You should not normally need to delete messages on a message point. This task is intended as part of a troubleshooting procedure.

To delete one or messages on a message point, use the administrative console to complete the following steps:

1. List the messages on the message point.
2. In the content pane, click the check box next to each message you want to delete. Alternatively, you can select all messages in the list by clicking the Select all items button 
3. Click **Delete**

Results

The selected messages are removed from the list.

Administering durable subscriptions

Use the following tasks to display the durable subscriptions that exist, to enable a subscription to be changed, or to delete a subscription.

About this task

The default messaging provider supports the use of durable subscriptions to topics. This enables a subscriber to receive a copy of all messages published to a topic, even messages published during periods of time when the subscriber is not connected to the server.

If an application creates a durable subscription, it is added to the list that administrators can display and act upon by using the administrative console. Each durable subscription is created with a unique subscription identifier, *clientID##subName* where:

clientID
: The client identifier used to associate a connection and its objects with the messages maintained for applications (as clients of the JMS provider). You should use a naming convention that helps you identify the applications, in case you need to relate durable subscriptions to the associated applications for runtime administration. For more information about client identifiers, see section 4.3.2 of the JMS 1.1 specification.

subName
: The JMS durable subscription name used to uniquely identify a durable subscription within a given client identifier. For more information about JMS durable subscription names, see section 6.11.1 of the JMS 1.1 specification.

For durable subscriptions created by message-driven beans, the subscription name value is set on the JMS activation specification. For other durable subscriptions, the value is set by the administrator on the JMS connection factory and by the JMS application on the createDurableSubscriber operation.

- Listing durable subscriptions
- Stopping active subscribers for durable subscriptions
- Enabling providers to stream messages to cloned durable subscriptions
- Deleting durable subscriptions

Listing subscriptions

Use this task to list the subscriptions that exist at runtime for a topic space.

About this task

The runtime state of a subscription is linked to the publication point for the messaging engine that is used to store messages delivered to the subscription. You can list subscriptions by first displaying the publication point, either from the messaging engine panel or the list of all publication points for a service integration bus.

To display a list of subscriptions for a publication point, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**. This displays any existing destinations in the content pane.
2. Click the name of the topic space.
3. Under Message Points, click **Publication points**. This displays the publication points for the topic space.
4. Click the name of the publication point.
5. Click the Runtime tab.
6. Under Additional Properties, click **Subscriptions**

Results

The list of subscriptions for the selected topic space is displayed in the content pane. For more information about the list of subscriptions, see Subscriptions [Collection].

Each subscription has a unique subscription identifier that has the form, *clientID##subName* where:

clientID
: The client identifier used to associate a connection and its objects with the messages maintained for applications (as clients of the JMS provider). You should use a naming convention that helps you identify the applications, in case you need to relate subscriptions to the associated applications for runtime administration. For more information about client identifiers, see section 4.3.2 of the JMS 1.1 specification.

subName
: The JMS subscription name used to uniquely identify a durable subscription within a given client identifier. For more information about JMS subscription names, see section 6.11.1 of the JMS 1.1 specification.

What to do next

To display details of a subscription, click the name of the subscription in the list. To delete one or more subscriptions, click the check box next to the subscription names then click **Delete**.

Stopping active subscribers for durable subscriptions

Use this task to enable applications to change a cloned durable subscription. This task enables applications to connect to an existing cloned durable subscription, but specify parameters that differ from those that were used to create the existing subscription.

About this task

When an application connects to an existing durable subscription, but specifies parameters that differ from those that were used to create the existing subscription, **the subscription is deleted then recreated with the new parameters**. A durable subscription can be changed in this way only when it has no active consumers.

To stop active subscribers for one or more durable subscriptions, complete the following steps:

1. Use the administrative console to list the durable subscriptions
2. From the list, identify the client identifier of the durable subscription. The name column lists the unique subscription name for each durable subscription, in the form *clientID##subName* where:

   clientID
   The client identifier used to associate a connection and its objects with the messages maintained for the client by the JMS provider.

   subName
   The name used to uniquely identify a durable subscription within a given client identifier.
3. Use your client identifier naming convention to identify the application assigned to the client identifier.
4. List the applications that have active consumers for the durable subscription. In the navigation pane of the administrative console, click **Applications** → **Application Types** → **WebSphere enterprise applications**.
5. In the console pane, select the check box next to the name of each application that you want to stop.
6. Click **Stop**

Results

This stops the active consumers created by the applications, so applications can reconnect to the durable subscriptions with different parameters from those that were used to create the previous subscriptions.

Deleting durable subscriptions

Use this task to delete a durable subscription.

About this task

To delete one or more durable subscriptions, complete the following steps:

1. Display the durable subscriptions
2. Select the check box for each subscription you want to delete.
3. Click **Delete**

Results

The selected durable subscriptions are deleted.

Mediations

These topics provide information about mediations, which are used to change how messages are handled at destinations on a service integration bus.

- Learning about mediations
- Learning about programming mediations
- Programming mediations
- “Securing mediations” on page 1254
- “Configuring mediations” on page 1256

- “Configuring mediation points” on page 1266
- “Managing mediations with administrative commands” on page 1268
- “Operating mediations at mediation points” on page 1268
- “Administering messages on mediation points” on page 1270
- Message properties support for mediations
- Error handling in mediations
- Mediation thread pool properties

Securing mediations

Use the following tasks to secure mediations at an operations level. For example, a mediation inherits its identity from a the messaging engine, but you might want to specify an alternative identity for the mediation to use.

Configuring an alternative mediation identity for a mediation handler

Use this task to configure an alternative mediation identity for a mediation handler

About this task

By default, a mediation inherits the identity used by the messaging engine. In some cases, you might want to specify an alternative identity for a mediation handler to use. For example, for a single mediation that sends messages to a destination. To do this, you specify a ″run-as″ identity for the mediation handler at deployment, and map the mediation handler to an identity other than the default mediation identity using a role name. Follow the steps below to specify an alternative mediation identity:

1. Package your mediation handler as an EAR file.
2. Edit the deployment descriptor file to define the roles. For more information, see Securing enterprise bean applications using the assembly tools.
3. Assign users to the role. For more information, see Mapping users to RunAs roles using an assembly tool and Assembling secured applications.
4. Deploy the mediation handler in WebSphere Application Server, and assign users to the RunAs role. For more information, see Assigning users to RunAs roles. You can confirm the mappings of users to roles, add new users and groups, and modify existing information during this step. For more information, see Deploying secured applications.

Example

What to do next

Next, you are ready to authorize mediations to access destinations. For more information, see “Administering authorization permissions” on page 1281.

Configuring the bus to access secured mediations

Use this task to ensure that the service integration bus is authorized to access secured mediations.

Before you begin

The mediation is secured using a Java Platform, Enterprise Edition (Java EE) Connector Architecture authentication alias. For information about creating a Java EE authentication alias, refer to Managing J2EE Connector Architecture authentication data entries.

About this task

To configure the bus to access a secured mediation, you must add the mediation authentication alias for the secured mediation to the properties for the bus:

- If the bus has a Version 6.x bus member, you must provide the principal and its associated password.
- If the bus has WebSphere Application Server Version 7.0 bus members only, you need only provide the principal.

1. Log into the navigation pane.
2. Click **Service integration** → **Buses** → ***security_value***. The bus security configuration panel is displayed.
3. In the **Mediations authentication alias** field, select the principal for the mediation, and its associated password if required.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

The selected bus is configured to access secured mediations.

What to do next

You can assign security roles to your mediation handlers to protect them from use by unauthorized users. For more information, see Deploying secured applications.

Configuring a bus to run mediations in a multiple security domain environment

Use this task to configure a secured bus so that it can run mediations successfully on bus members in different security domains.

Before you begin

The secured bus must be configured to use a non-global security domain. For more information about securing buses using multiple security domains, refer to “Securing buses” on page 1272.

About this task

If your bus topology has bus members in different security domains, you can configure the bus to allow mediations to run under the server identity. This means that a mediation can run on any server in any domain. You do not have to add a dedicated user ID for each mediation to the user repository, or maintain a mediation authentication alias.

Use the administrative console to configure a secured bus to run mediations successfully as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The security settings for the selected bus are displayed.
2. Check the option **Use the Server ID when running mediations**.
3. Click **Apply**.
4. Save your changes to the master configuration.

Results

You have configured the bus to run mediations successfully across servers in multiple security domains.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Configuring mediations

Use the following tasks if you want to modify the behavior of a mediation, control which messages are mediated, or influence how messages are processed.

Installing a mediation into WebSphere Application Server

This task describes how to make a mediation available to destinations on a service integration bus. The mediation is an enterprise application that you can deploy on WebSphere Application Server.

Before you begin

- You have an enterprise application (EAR file) that you can deploy into WebSphere Application Server (base). The EAR file contains a mediation handler enterprise bean. You can use the tooling provided with IBM Rational Application Developer to assemble EAR files.
- For guidance on selecting the target application server for your mediation, see Considerations for installing a mediation.

About this task

In this task, you use the administrative console to install a mediation application into WebSphere Application Server (base). For additional information about installing enterprise applications, see Installing application files with the console. For information about installing a secure mediation handler, see Deploying secured applications.

1. Log into the administrative console.
2. Click **Applications** → **New Application** → **New Enterprise Application**.
3. Prepare to install the mediation handler application:
   a. Specify the file path for the mediation handler application.
   b. Select **Detailed - Show all installation options and parameters**.
   c. Expand **Choose to generate default bindings and mappings**.
   d. Select **Generate default bindings and mappings**.
   e. Click **Next**.
4. Type the name of the mediation handler in the **Application name** field, if not already specified, and click **Next**.
5. Select the target application server where you want to install the modules in the mediation handler application, and click **Next**.
6. Optional: If the panel **Deploy enterprise beans** is displayed, click**Apply**, and then click **Next**.
7. Click **Finish**.
8. Save your changes to the master configuration.

Results

The mediation application is installed on the target WebSphere Application Server.

What to do next

Create the mediation in the administrative console. For more information, see “Defining a mediation.”

Defining a mediation

Use this task to define a mediation in the administrative console.

Before you begin

This task assumes that you have installed your mediation handler module on the WebSphere Application Server server where you intend to use your mediation. For more information, see “Installing a mediation into WebSphere Application Server” on page 1256.

About this task

Defining a mediation in the administrative console makes it available for the administrator to use to mediate destinations.

To define a mediation, use the administrative console to complete the following steps:

1. Display the **Mediations** page. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations**
2. In the content pane, click **New**.
3. Specify the following properties for the mediation:

   Mediation name
   Type a name for the mediation that is unique to the service integration bus. This name is used to identify the mediation for administrative purposes.

   Description
   Optionally, type a description of the behavior of the mediation, taking into account any properties specified for the mediation.

   Handler list name
   Type the handler list name. This is the name used by the programmer who deployed the mediation handler. Handler list names are unique within the WebSphere Application Server administrative cell.

   Global transaction
   Optionally, select this option (setting it to true) if you want to start a global transaction for each message mediated by the mediation. By default, Global transaction is set to false, and a global transaction is not started when a message is mediated.

   Allow concurrent mediation
   Optionally, select this option (setting it to true) if you want to mediate multiple messages at the mediated destination.

   By default, Allow concurrent mediation is set to false, and concurrent mediations are not allowed.

   Note: Do not allow concurrent mediation if message ordering is important.

   Selector
   Optionally, type a message selector to control which messages are mediated by the mediation. The selector operates on the content of the message header. The syntax of the message selector is defined by the JMS specification.

   If the message meets the conditions of the selector, the message is mediated. Otherwise, the message is not mediated.

   Discriminator
   Optionally, type a message discriminator to control which messages are mediated by the mediation. Message discriminators conform to the publish/subscribe topicspace syntax.

   If the message meets the conditions of the discriminator, the message is mediated. Otherwise, the message is not mediated.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

A named mediation is defined for the deployed handler list.

What to do next

Next, you can add the mediation to a destination, see “Mediating a destination” on page 1264.

Deleting a mediation

Use this task to delete a selected mediation in the administrative console.

Before you begin

Before deleting a mediation, you must ensure that it is not in use at a bus destination. You cannot delete a mediation that is in use at a destination; you must first unmediate the destination. For more information, see “Unmediating a destination” on page 1265.

About this task

Deleting a mediation removes it from the administrative console. To delete a mediation, use the administrative console to complete the following steps

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations**.
2. Select the mediations that you want to delete.
3. Click **Delete**.
4. Save your changes to the master configuration.

Results

The mediations are deleted.

Configuring mediation properties

Use this task to configure the properties of a mediation in the administrative console.

Before you begin

This task assumes that you have defined a new mediation. For more information, see “Defining a mediation” on page 1256.

About this task

Configuring the properties of a mediation controls how it behaves at run-time. To configure the properties of a selected mediation, use the administrative console to complete the following steps.

1. Display a selected mediation. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations** → ***mediation_name***.
2. Specify the following properties for the mediation:

   Description
   Optionally, type a description of the behavior of the mediation, taking into account any properties specified for the mediation.

   Handler list name
   Type the name of the handler list that was defined when the mediation was deployed. Handler list names start with an uppercase letter, and are unique within the service integration bus.

Global transaction
Optionally, select this option (setting it to true) if you want to start a global transaction for each message mediated by the mediation.

By default, Global transaction is set to false, and a global transaction is not started when a message is mediated.

Allow concurrent mediation
Optionally, select this option (setting it to true) if you want to mediate multiple messages at the mediated destination.

By default, Allow concurrent mediation is set to false, and concurrent mediations are not allowed.

Note: Do not allow concurrent mediation if message ordering is important.

Selector
Optionally, type a message selector to control which messages are mediated by the mediation. The selector operates on the content of the message header. The syntax of the message selector is defined by the JMS specification.

If the message meets the conditions of the selector, the message is mediated. Otherwise, the message is not mediated.

For information about the properties that can be used in selectors, see Message properties support for mediations.

Discriminator
Optionally, type a message discriminator to control which messages are mediated by the mediation. Message discriminators conform to the publish/subscribe topic space syntax.

If the message meets the conditions of the discriminator, the message is mediated. Otherwise, the message is not mediated.

3. Specify any additional properties.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

The configuration properties for the mediation are created.

What to do next

If you want to configure the mediation context information, see “Configuring mediation context properties” on page 1261. To use the configured mediation at a destination, see “Mediating a destination” on page 1264.

Adding mediation context information

Use this task to add a mediation context property for a selected mediation.

Before you begin

The mediation for which you want to add context information must exist in the administrative console. To create a mediation, see “Defining a mediation” on page 1256.

About this task

Mediation context information is used, in addition to the message header information, to determine how a message is processed at run time. New context information is defined by adding a mediation context

property. A destination can also have context properties. If you add a context property for a mediation that has the same name as a destination context property, the property on the destination takes precedence.

You can add the following mediation context properties:

Property name	Description	Value	Additional information
sib:SkipWellFormedCheck	Omits the well formed check that is performed on messages after they have been processed by the mediation.	True or false	This property is overridden by the message delivery option assured persistent. A well formed check is always performed for messages that have the delivery option assured persistent.
sib:GlobalTransactionLPSEnabled	Allows a mediation to contain resources for which Last Participant Support (LPS) is enabled. This enables a one phase commit resource to participate in a global transaction that has two phase commit resources.	True or false	

To add new mediation context information for a selected mediation, use the administrative console to complete the following steps:

1. Display the Context Properties panel for a selected mediation: click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations** → ***mediation_name*** → **[Additional Properties] Context properties**. Any mediation context properties that have already been defined for the selected mediation are displayed.
2. Click **New** in the content pane.
3. Specify the following information for the mediation context property:

   Name Type a name for the mediation context property. You must type the name exactly as it appears in the table above.

   Context type
   Select the type `Boolean` for the mediation context property.

   Context value
   Type `True` to set the property. The value is not case sensitive.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

You have added new mediation context information for the selected mediation.

What to do next

You can repeat this task to add another mediation context property.

Listing mediation context properties

Use this task to list context properties for a selected mediation.

About this task

Mediation context information is used, in addition to the message header information, to determine how a message is processed at run time. The context information is provided by one or more of the following mediation context properties:

Property name	Description	Value	Additional information
sib:SkipWellFormedCheck	Omits the well formed check that is performed on messages after they have been processed by the mediation.	True or false	This property is overridden by the message delivery option assured persistent. A well formed check is always performed for messages that have the delivery option assured persistent.
sib:GlobalTransactionLPSEnabled	Allows a mediation to contain resources for which Last Participant Support (LPS) is enabled. This enables a one phase commit resource to participate in a global transaction that has two phase commit resources.	True or false	

In this task, you use the administrative console to list the mediation context properties for a selected mediation, as follows:

1. Log in to the administrative console.
2. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations** → ***mediation_name*** → **[Additional Properties] Context properties**.

Results

The mediation context properties for the selected mediation are displayed.

What to do next

You can configure or delete the context information, or add a new context property for the selected mediation.

Configuring mediation context properties

Use this task to configure context properties for a selected mediation.

Before you begin

The mediation context property you want to configure must exist in the administrative console. To add a new mediation context property, see “Defining a mediation” on page 1256.

About this task

Mediation context information is used, in addition to the message header information, to determine how a message is processed at run time. By configuring the mediation context properties, you ensure that messages are processed correctly. A destination can also have context properties. If a context property is configured for both a destination and a mediation, with the same name for the property, the property on the destination takes precedence.

You can configure the following mediation context properties:

Property name	Description	Value	Additional information
sib:SkipWellFormedCheck	Omits the well formed check that is performed on messages after they have been processed by the mediation.	True or false	This property is overridden by the message delivery option assured persistent. A well formed check is always performed for messages that have the delivery option assured persistent.

Property name	Description	Value	Additional information
sib:GlobalTransactionLPSEnabled	Allows a mediation to contain resources for which Last Participant Support (LPS) is enabled. This enables a one phase commit resource to participate in a global transaction that has two phase commit resources.	True or false	

To configure an existing mediation context property, use the administrative console to complete the following steps:

1. Display the mediation context property for a selected mediation: click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations** → ***mediation_name*** → **[Additional Properties] Context properties**. The mediation context properties for the selected mediation are displayed.
2. Select the mediation context property you want to configure.
3. Specify the following information for the mediation context property:

   Name Type a name for the mediation context property. You must type the name exactly as it appears in the table above.

   Context type
   Select the type `Boolean` for the mediation context property.

   Context value
   Type a value for the mediation context property: either `True` to set the property, or `False` to unset it. The value is not case sensitive.
4. Click **OK**.
5. Save your changes to the master configuration.

Results

The context property is configured for the selected mediation.

What to do next

You can repeat this task to configure additional mediation context properties.

Deleting mediation context information

Use this task to delete a mediation context property for a selected mediation.

About this task

In this task, you delete a mediation context property. The mediation context information provided by the property is no longer used at runtime to process messages.

You can delete the following mediation context properties:

Property name	Description	Value	Additional information
sib:SkipWellFormedCheck	Omits the well formed check that is performed on messages after they have been processed by the mediation.	True or false	This property is overridden by the message delivery option assured persistent. A well formed check is always performed for messages that have the delivery option assured persistent.

Property name	Description	Value	Additional information
sib:GlobalTransactionLPSEnabled	Allows a mediation to contain resources for which Last Participant Support (LPS) is enabled. This enables a one phase commit resource to participate in a global transaction that has two phase commit resources.	True or false	

To delete mediation context information for a selected mediation, use the administrative console to complete the following steps:

1. Display the Context Properties panel for a selected mediation: click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations** → ***mediation_name*** → **[Additional Properties] Context properties**. The mediation context properties for the selected mediation are displayed.
2. Select the property you want to delete, and click **Delete**.
3. Click **OK**.
4. Save your changes to the master configuration.

Results

Your chosen mediation context property is deleted for the selected mediation.

What to do next

You can repeat this task to delete another mediation context property.

Configuring the mediation thread pool

Use this task to configure the mediation thread pool.

Before you begin

The wsadmin tool must be running. For more information, see Starting the wsadmin scripting client.

About this task

You need to configure the mediation thread pool if you want to change the number of threads used when running mediations concurrently. The maximum size of the thread pool determines the maximum number of messages that can be mediated concurrently for a messaging engine.

The mediations thread pool, attribute name `mediationsThreadPool`, is an attribute of the messaging engine. By default, `mediationsThreadPool` does not exist, and a default thread pool is created and used at run time. In this task, use the wsadmin tool to create a thread pool object, and then modify its properties using JACL, as shown in the examples below:

Note: You use the wsadmin tool from within Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.

1. Use this example to create a mediationThreadPool object:

```
AdminConfig.create("ThreadPool" , messagingEngine,
                   [["name" , "stitch.server1-bus2-mediationThreadPool"]] ,
                   "mediationThreadPool")
```

   Note: In this case, the thread pool name is based on the name of the messaging engine. Although it is not required to do this, it makes it easier to find the thread pool name when using Performance Monitoring Infrastructure (PMI).

2. Use this example to modify a mediationThreadPool object:

```
AdminConfig.modify(AdminConfig.showAttribute(messagingEngine,
  "mediationThreadPool"), [["maximumSize" , "10"]])
```

`maximumSize` can contain any of the mediationsThreadPool properties. To add additional parameters, insert [*attribute_name attribute_value*] within the outer brackets ([]).

Note: There is a space between *attribute_name* and *attribute_value*.

Setting tuning properties for a mediation

Use this task to tune a mediation for performance by using the administrative console.

Before you begin

Review the guidance on when it is appropriate to tune a mediation for performance in the topic Guidance for tuning mediations for performance.

About this task

You can set the following tuning property in the administrative console to improve the performance of a mediation:

sib:SkipWellFormedCheck
Whether you want to omit the well formed check that is performed on messages after they have been processed by the mediation. Either true or false.

Note: This property is overridden for messages that have the delivery option assured persistent, and a well formed check is always performed.

To set, or unset, one or more tuning properties for a mediation, use the administrative console to complete the following steps:

1. Display the mediation context information:
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations**.
   b. In the content pane, select the name of the mediation for which you want to configure tuning information.
   c. Click **[Additional Properties] Context information**.
2. In the content pane, click **New**.
3. Type the name of the property in the **Name** field.
4. Select the type `Boolean` in the list box.
5. Type **true** in the **Context Value** field to set the property, or type **false** to unset the property.
6. Click **OK**.
7. Save your changes to the master configuration.

Mediating a destination

Use this task to mediate a selected bus destination.

Before you begin

This task assumes that your mediation is installed and configured on the same bus member as the destination you want to mediate. For more information, see “Defining a mediation” on page 1256.

About this task

mediations

Mediating a destination associates a mediation with a selected bus destination. At run time, the mediation applies some message processing to the messages handled by the bus destination. Note that you can only mediate a destination with a single mediation at a time. You can mediate more than one destination with the same mediation. To mediate a destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**. This displays a list of bus destinations.
2. In the content pane, select the bus destination you want to mediate, and click **Mediate**.
3. Select the mediation to associate with the destination, and click **Next**.
4. Select the bus member where the mediation is installed, and click **Next**.
5. Click **Finish** to mediate the destination, or **Previous** to make any changes to your selections.
6. Save your changes to the master configuration.

Results

The destination is mediated, and a mediation point is created for the destination on the bus member.

What to do next

Next, you can configure properties that affect how messages are processed at the mediated destination; for example, whether it runs within a global transaction, or processes multiple messages concurrently. For more information, see “Configuring mediation properties” on page 1258.

Unmediating a destination

Use this task to unmediate a selected bus destination.

Before you begin

Before you start this task, you should consider whether you want to preserve message ordering at the bus destination you intend to unmediate.

About this task

Unmediating a selected bus destination breaks the association between the bus destination and the mediation, and the mediation stops processing messages at the bus destination at run time. This task unmediates a bus destination, and optionally preserves message ordering at the bus destination.

To unmediate a bus destination use the administrative console to complete the following steps:

Note: If you want to preserve message ordering, complete all the following steps. If message ordering is not important, you can omit steps 2 and 4:

1. Display the bus destination you want to unmediate. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
2. Optional: If you want to preserve message ordering at the destination, take the following steps:
   a. Clear the **Send Allowed** check box (setting it to false) to stop sending messages to the message points for this destination.
   b. Wait until there are no messages listed for the mediation points.
   c. Stop all the mediation points, and wait for all the mediation points to enter the `stopped` state.
   d. Optional: If it is not already selected, select the bus destination you want to unmediate.
3. Click **Unmediate** to unmediate the destination.
4. Optional: Select the **Send Allowed** check box (setting it to true) to resume sending messages to the mediation points for this destination.

5. Save your changes to the master configuration.

Results

The destination is unmediated, and the mediation point is deleted from the destination.

If you did not preserve message ordering, any messages that remain on the mediation point after you have unmediated the destination are processed as for a non-mediated destination. Messages with an empty forward routing path are placed on the next appropriate message point for the destination. Messages with a non-empty forward routing path are sent to the destination specified in the forward routing path.

The unmediated destination now operates as a non-mediated destination, and new messages are sent to a message point on the destination.

Configuring mediation points

In service integration technologies, messages are held at specialized message points called mediation points before they are processed by a mediation. Use the following tasks to set properties at mediation points to route messages and start and stop mediations:

Configuring a mediation point

This task uses the administrative console to browse or change the configuration properties for a selected mediation point for a selected bus destination.

Before you begin

You must first create a mediation point on a bus destination. For more information, see “Mediating a destination” on page 1264.

About this task

A mediation point is a specialized message point on a bus location where messages are stored and mediated. Configuring the properties of a mediation point enables you to control some aspects of how messages are sent to and received from a bus destination. In this task you use the administrative console to configure the properties for a selected mediation point for a selected bus destination.

Note: You should be aware that the mediation point properties Send allowed and Initial state override the general properties defined for the bus destination.

To browse or change the properties for a selected mediation point, use the administrative console to complete the following steps:

1. Start the administrative console, and click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points** → ***mediation_point_name***. The properties for the selected mediation point are displayed.
2. Optional: Browse or change the following properties for the mediation point:

   Identifier
   The identifier by which this message point is known for administrative purposes. This field is for display purposes only.

   UUID The universal unique identifier assigned to this mediation point for administrative purposes. This field is for display purposes only.

   High message threshold
   The maximum total number of messages that can be placed on this mediation point. This field is for display purposes only.

Send allowed
This property specifies whether or not messages are sent to this mediation point. By default, `True` is selected, and messages are sent to this mediation point. Select `False` to prevent messages from being sent to this mediation point.

Initial state
This property specifies the state of the mediation point when the host messaging engine is started for the first time. Permitted values are `Started` or `Stopped`. The default state is `Started`. The mediation does not start mediating messages until Initial state is `Started`.

Target UUID
The UUID of the bus destination where this mediation point is created. This field is for display purposes only.

3. Click **OK**.
4. Save your changes to the master configuration.

Results

You have browsed or changed the configuation properties for the selected mediation point.

Listing mediation points for a bus destination

Use this task to list mediation points for a selected bus destination.

About this task

This task lists the mediation points for a selected bus destination. You might want to do this, for example, to check that a destination is mediated.

To display a list of mediation points for a bus destination, use the administrative console to complete the following steps:

1. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name***.
2. Click **[Message Points] Mediation Points**.

Results

All the mediation points for the selected bus destination are displayed.

Listing mediation points for a messaging engine

This task displays a list of mediation points for a selected messaging engine.

About this task

This task uses the administrative console to list all the known mediation points for a selected messaging engine. You can then view the properties for a mediation point, administer messages for a mediation point, or operate mediations at a mediation point.

To list the mediation points for a selected messaging engine, use the administrative console to complete the following steps:

Start the administrative console, and click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points**.

Results

The Mediation points panel displays a list of all the mediation points for the selected messaging engine.

What to do next

You can click the name of a mediation point in the list to work with its properties, administer its messages or operate mediations.

Managing mediations with administrative commands

You can use wsadmin commands to administer service integration technologies mediations. For example you can create, delete and view mediations, configure mediation properties and mediate destinations.

About this task

These commands provide an alternative to using the administrative console.

1. Open a wsadmin command session in local mode For example:

   ```
   wsadmin -conntype none -lang jython
   ```

   Note: The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.

2. Type `AdminTask.`*`command`* where *command* is the command format as indicated in the related reference topics. For example:

   ```
   wsadmin>AdminTask.listSIBMediations("-bus abus")
   (cells/9994GKCCell01/buses/abus|sib-mediations.xml#SIBDestinationMediation_1098217858584)
   (cells/9994GKCCell01/buses/abus|sib-mediations.xml#SIBDestinationMediation_1098220051588)
   ```

Operating mediations at mediation points

Use these tasks to start, stop and restart mediations at runtime at specialized message points called mediation points.

Before you begin

These tasks assume that mediations are configured in the administrative console. For more information, see “Configuring mediations” on page 1256.

Starting a mediation

Use this task to start a selected mediation at a mediation point.

Before you begin

Before you can start a mediation, it must be associated with a destination. For more information, see “Mediating a destination” on page 1264.

About this task

You start a mediation when you want the mediation to start mediating messages. Starting a mediation, like all runtime operations on mediations, is carried out at a specialized message point on the destination, called a mediation point. A mediations starts operating on messages after the server has started and no new messages are mediated after server shutdown begins.

Note: If you want to ensure that the mediation starts processing a specific message, remove existing messages from the mediation point before you start the mediation. For more information, see “Deleting messages on a mediation point” on page 1270.

To start a mediation, use the administrative console to complete the following steps:

1. List the mediation points.

- To list the mediation points for a bus destination, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name*** → **[Message Points] Mediation points**.
- To list the mediation points for a messaging engine, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points**.

2. Select the mediation name.
3. Click **Start**.

Results

The mediation starts processing messages mediation point that arrive at the mediation point.

Stopping a mediation

Use this task to stop a selected mediation at a mediation point.

Before you begin

Before you can operate a mediation, it must be associated with a destination. For more information, see "Mediating a destination" on page 1264.

About this task

Stopping a mediation, like all runtime operations on mediations, is carried out at a specialized message point on the destination, called a mediation point. By stopping a mediation, you stop any further messages from being delivered to the mediation point. Messages sent to the destination are held at the pre-mediated part of the destination. The messages are not mediated and are not made available to message consumers until the mediation is restarted. Note that mediations starts operating on messages after the server has started and no new messages are mediated after server shutdown begins.

To stop a selected mediation, use the administrative console to complete the following steps:

1. List the mediation points.
   - To list the mediation points for a bus destination, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name*** → **[Message Points] Mediation points**.
   - To list the mediation points for a messaging engine, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points**.
2. Select the mediation name.
3. Click **Stop**.

Results

The mediation is stopped at the mediation point. Messages arriving at the mediation point are not processed by the mediation. The messages are stored at the mediation point until the mediation is restarted.

Restarting a mediation that has stopped on error

This topic describes how to restart a mediation that has stopped on error.

About this task

You need to restart a mediation that has stopped on error so that the mediation can continue processing messages. When the mediation is restarted, the mediation resumes mediating messages on the pre-mediated part of the destination.

Changes in the state of a mediation normally occur as the result of operations performed by the administrator. If the state of a mediation moves from `Started` to `Stopped on error`, and you have not stopped it intentionally, you should investigate the cause of the problem, and resolve it before attempting to restart the mediation.

To restart a mediation that has stopped on error, use the administrative console to complete the following steps:

1. List the mediation points.
   - To list the mediation points for a bus destination, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name*** → **[Message Points] Mediation points**.
   - To list the mediation points for a messaging engine, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points**.
2. In the content pane, select the mediation name.
3. Click the **Runtime** tab.
4. Under **General Properties**, view the information in the **Reason** field.
5. Resolve the problem.
6. Restart the mediation so that it can continue processing messages.

Administering messages on mediation points

Use these tasks to list and delete runtime messages held at mediation points in a service integration bus.

Listing messages at a mediation point

Use this topic to list messages at a mediation point.

About this task

The mediation point is a specialized message point that defines the association between a mediation and a destination. You can browse and delete messages on a mediation point for a selected bus destination or messaging engine.

To display a list of messages on a selected mediation point, use the administrative console to complete the following steps:

1. List the mediation points.
   - To list the mediation points for a bus destination, click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***destination_name*** → **[Message Points] Mediation points**.
   - To list the mediation points for a messaging engine, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Message points] Mediation points**.
2. Click the mediation point name. This displays the properties of the mediation point.
3. Click the **Runtime** tab.
4. Click **[Additional Properties] Messages**.

Results

A list of messages on the selected mediation point is displayed in the content pane.

Deleting messages on a mediation point

Use this topic to delete messages that exist on a mediation point for a selected bus destination.

About this task

The mediation point is a specialized message point that defines the association between a mediation and a destination. Like other message points, you can delete messages on a mediation point for a selected bus destination or messaging engine. For example, if you want to preserve message ordering at a destination before it is unmediated, you can delete all the messages at that destination.

1. List the messages at the mediation point. For more information, see “Listing messages at a mediation point” on page 1270.
2. In the content pane, click the check box next to each message you want to delete. Alternatively, select all the messages in the list by clicking **Select All Items**.
3. Click **Delete**.

Results

The selected messages are removed from the destination.

Example: Using mediations to trace, monitor and log messages

The most straightforward use of a mediation is for tracing, monitoring or logging messages that pass through a destination or topics spaces. This type of mediation does not modify the message; it simply extracts information from the message and saves or displays the information elsewhere.

For example, the following mediation handler displays the API message and correlation IDs for each message it is sent:

```
public boolean(MessageContext context)
{
  SIMessageContext   msgCtx  = (SIMessageContext)context;
  SIMediationSession session = msgCtx.getSession();
  SIMessage          msg     = msgCtx.getMessage();
  String             msgId   = msg.getApiMessageId();
  String             corrId  = msg.getCorrelationId();
  String             dest    = session.getDestinationName();

  System.out.println(msgId+" (correlation id="+corrid) is passing through "+dest+".");

  return true;
}
```

Security

This topic is an overview of the tasks for administering messaging security for a service integration bus.

Before you begin

Review the security requirements for the bus. For guidance, see Planning your security requirements.

About this task

Messaging security protects the bus from unauthorized access. By default, messaging security is enabled for the bus. Providing administrative security is also enabled, messaging security enforces a security policy that prevents unauthorized client applications from connecting to the bus, and accessing bus resources. There may be circumstances when you do not require messaging security, for example on a development system. In this case, you can disable messaging security.

You can customize the security configuration for the bus using the administrative console, or wsadmin scripting commands. The security configuration controls the following aspects of bus security:

- Authorizing groups of users in the user registry to perform selected operations on bus destinations.
- The transport policies that maintain the integrity of messages in transit on the bus.
- The use of global, and multiple custom security domains.
- The integrity of links between messaging engines, foreign buses and databases.

Use the following tasks to administer messaging security:
- “Securing buses”
- “Enabling client SSL authentication” on page 1279
- “Adding unique names to the bus authorization policy” on page 1281
- “Administering authorization permissions” on page 1281
- “Administering permitted transports for a bus” on page 1307
- “Securing messages between messaging buses” on page 1311
- “Securing access to a foreign bus” on page 1312
- “Securing links between messaging engines” on page 1313
- “Controlling which foreign buses can link to your bus” on page 1314
- “Securing database access” on page 1314
- “Securing mediations” on page 1254

Securing buses

Securing a service integration bus provides the bus with an authorization policy to prevent unauthorized users from gaining access. If a bus is configured to use multiple security domains, the bus also has a security domain and user realm to further enforce its authorization policy.

Before you begin

- If administrative security for the cell in which the bus resides is not already enabled, you must enable it. The tasks below use an administrative console wizard that detects if administrative security is not enabled, and takes you through the steps to enable it. You need to supply the type of user repository used by the server, and the administrative security username and password.
- If the bus contains a bus member at WebSphere Application Server Version 6.x, you must provide an authentication alias to establish trust between bus members, and to enable the bus to operate securely. The administrative console wizard detects if an authentication alias is required, and prompts you to supply one. If you want to specify a new authentication alias, you must provide a username and password.

About this task

When you secure a bus, consider the following points:

- If you are securing a WebSphere Application Server Version 7.0 bus that contains only Version 7.0 bus members, you can use a non-global security domain for the bus. If the bus has a WebSphere Application Server Version 6.x bus member, or might have a Version 6.x bus member in the future, you must assign the bus to the global security domain.
- If you want to assign the bus to a custom domain, you can select an existing security domain, or create a new one.
- If you assign the bus to a custom domain, you must specify a user realm. You can select an existing user realm, or use the global user realm.

What to do next

- The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.
- Use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Disabling bus security

If you do not want to protect the bus from unauthorized access, you can disable messaging security.

Before you begin

- Ensure that there are no indoubt transactions on the messaging engine because incomplete transactions cannot be recovered after the bus security is disabled. For more information, refer to Resolving indoubt transactions.
- Stop all servers on which the SIB Service enabled before you disable bus security. This ensures that the bus security configuration is applied consistently when the servers are restarted. For more information, refer to Stopping an application server.

About this task

Messaging security protects the bus from access by unauthorized client applications. Messaging security is enabled by default, providing administrative security for the cell is enabled. However, there may be circumstances when you do not require messaging security, for example on a development system. In this case, you can use the administrative console to disable messaging security. If you want to re-enable bus security, refer to “Securing buses” on page 1272.

1. In the navigation pane, click **Service integration** → **Buses**. A list of buses is displayed.
2. Find the bus for which you want to disable security, and click **Enabled** in the security column. The security settings for the selected bus are displayed.
3. Clear the checkbox **Enable bus security**.
4. Click **Apply**.
5. Save your changes to the master configuration.

Results

You have disabled security for the selected bus.

What to do next

You must restart the servers. Starting an application server provides more information.

Adding a secured bus

This task uses an administrative console wizard to add a new service integration bus that uses a security domain. A bus can use the global security settings, inherit the security settings that exist at cell level, or use a set of customized settings that are unique to the bus, or shared with another resource.

Before you begin

You must decide which type of security domain to use for the bus. For more information, refer to Planning your security requirements and Messaging security and multiple security domains.

About this task

By default, administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, the wizard prompts you to enable it. You must specify the type of user repository, and the administrative security username and password.

Security settings are held in a security domain. If the bus might contain a WebSphere Application Server Version 6.x bus member, you must select the global domain. You can select any type of domain for a bus that contains only WebSphere Application Server Version 7.0 bus members.

1. Log into the administrative console.
2. Click **Service integration** → **Buses**. A list of buses is displayed.

3. In the content pane, click **New**.
4. Type a name for the new bus. It is advisable to choose bus names that are compatible with the WebSphere MQ queue manager naming restrictions. You cannot change a bus name after a bus is created, so if you need to interoperate with WebSphere MQ in the future, it will be much simpler if you use compatible names. See the topic about WebSphere MQ naming restrictions in the related links.
5. Ensure that the **Bus security** checkbox is checked.
6. Click **Next**. The Bus Security Configuration wizard is launched.
7. Read the Introduction panel, and click **Next**.
8. If administrative security is disabled, follow the instructions in the wizard to select, and configure the appropriate user repository.
9. Click **Next**. A summary of the administrative security settings for the bus is displayed.
10. Review the summary, and click **Finish**. Administrative security for the cell is now enabled.
11. By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** .
12. Select a security domain for the bus. The domain you specify depends on the versions of the bus members. If the bus may contain a Version 6.x bus member, you must select the global domain. If the bus will only contain Version 7.0 bus members, you can select the cell-level or a custom domain. If you select a custom security domain, you must also follow the instructions to specify a user realm.
13. Review the summary of your choices, and click **Finish**.
14. Save your changes to the master configuration.

Results

Administrative security is enabled, and a new bus is created with your chosen security settings.

What to do next

You can now add bus members to the bus.

Securing an existing bus using multiple security domains

You can configure an existing service integration bus so that it uses a security domain other than the default global security domain. Using non-global security domains provides the scope to use multiple security domains. The bus can inherit security settings from the cell, or have a unique security configuration.

Before you begin

- For more information about using security domains, refer to Planning your security requirements and Messaging security and multiple security domains.
- The bus you want to secure must exist in the administrative console. If you want to create a new secured bus, refer to “Adding a secured bus” on page 1140.
- All the bus members must be at WebSphere Application Server Version 7.0; use of multiple security domains is not supported for earlier versions of WebSphere Application Server. If the bus you want to secure has a WebSphere Application Server Version 6.x bus member, refer to “Securing an existing bus using the global security domain” on page 1275.

About this task

This task uses the Bus Security Configuration wizard to secure a selected bus, and assign it to a security domain. If administrative security for the cell is disabled, the wizard prompts you to enable it. You need to know the type of user repository, and the administrative security username and password. If you specify a custom domain, you must also specify a user realm.

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The general properties for the selected bus are displayed.
2. Click **Configure Bus Security** to launch the Bus Security Configuration wizard.
3. Read the Introduction panel, and click **Next**.
4. If administrative security is disabled, follow the instructions to configure the appropriate user repository, and click **Next**.
5. Review the summary of your choices:
   a. If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
   b. Click **Finish** when you are ready to confirm your choices.

   Administrative security for the cell is now enabled.
6. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** . By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity.
7. Select the cell-level or custom security domain for the bus. You can only select a non-global security domain if all the bus members are at Version 7.0.
8. Optional: To create a new custom security domain:
   a. Use the name suggested for the security domain, or type a new one.
   b. Optional: Provide a description of the security domain.
   c. Select a user realm for the domain. You can use the user realm configured in the global security domain, or follow the steps to configure a new user realm.
9. Click **Next**.
10. Review the summary of your choices:
    a. Optional: If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
    b. Click **Finish** to confirm your choices.
11. Save your changes to the master configuration.

Results

You have specified that the selected bus uses a non-global security domain. The security settings configured for the bus are displayed in the updated Bus Security Settings panel. The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Securing an existing bus using the global security domain

Use this task to secure an existing service integration bus using the global security domain.

Before you begin

- The bus you want to secure must exist in the administrative console. If you want to create a new secured bus, refer to “Adding a secured bus” on page 1140.
- If administrative security for the cell in which the bus resides is disabled, the wizard prompts you to enable it. You need to know the type of user repository, and the administrative security username and password.

- If the service bus contains a bus member at WebSphere Application Server Version 6.x, the wizard prompts you to select an existing authentication alias, or specify a new one. If you want to specify a new authentication alias, you must provide a username and password.

About this task

Use this task if you want to secure a bus that exists already in the administrative console, and you want to use the global (default) security domain. For example, you are securing a Version 7.0 bus that has a bus member at WebSphere Application Server Version 6.x; mixed version buses cannot use multiple security domains.

This task uses an administrative console wizard to guide you through the steps to secure a bus. The following steps are conditional, depending on the bus environment:

- If administrative security for the cell in which the bus resides is disabled, the wizard prompts you to enable administrative security.
- If the bus has a bus member at WebSphere Application Server Version 6.x, the wizard prompts you for an authentication alias to establish trust between bus members, and to enable the bus to operate securely.

Use the administrative console to secure a selected bus using the global security domain as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The general properties for the selected bus are displayed.
2. Click **Configure Bus Security** to launch the Bus Security Configuration wizard.
3. Read the Introduction panel, and click **Next**. The next step is conditional, depending on whether administrative security is enabled or disabled:
   - If administrative security is disabled, complete all the following steps.
   - If administrative security is already enabled, continue from step 7.
4. Select the appropriate user repository, and click **Next**.
5. Depending on the type of user registry you selected, do one of the following:
   - For a federated repository, specify a username and password for administrative security, and click **Next**.
   - For all other types of repository, follow the wizard prompts, and click **Next**.
6. Review the summary of your choices:
   a. If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
   b. Click **Finish** when you are ready to confirm your choices.

   Administrative security for the cell is now enabled.
7. If you do not want clients to use SSL protected transports, clear the checkbox **Require clients use SSL protected transports** . By default, clients are required to use SSL protected transports to ensure data confidentiality and integrity.
8. Select the global security domain option, and click **Next**.
9. If at least one bus member is at Version 6.x, you must specify an authentication alias. Specify either an existing authentication alias, or create a new one:
   - Select **Specify existing authentication alias**, and select the alias name from the list box.
   - Select **Create a new authentication alias**, type a unique alias name and password.
10. Review the summary of your choices:
    a. Optional: If you want to make changes, click **Previous** to return to an earlier panel, and make the changes you require.
    b. Click **Finish** to confirm your choices.
11. Save your changes to the master configuration.

Results

You have secured the bus using the global security domain. The new security settings for the bus are displayed in the updated Bus Security Settings panel. The bus is secured after you restart all the servers that are members of the bus, or in the case of a bus that has bootstrap members, servers for which the SIB service is enabled.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Configuring bus security using an administrative console panel

Use this task to enable or disable messaging security, and configure security properties for an existing service integration bus.

Before you begin

The bus must exist in the administrative console. If you want to create a new bus, refer to “Adding buses” on page 1140.

About this task

This task uses the Bus Security administrative console panel. You can launch the Bus security wizard from the panel, or specify individual security properties directly in the panel. The bus security properties are effective only when administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, it prompts you to enable it.

The security properties available to a particular bus depend on the versions of the bus members. If the bus has a WebSphere Application Server Version 6.x bus member, you must specify the global security domain. The bus cannot use cell level, or custom security domains. You must also specify an inter-engine authentication alias to prevent unauthenticated messaging engines from establishing a connection with the bus. If the bus contains Version 7.0 bus members only, you can specify any type of security domain, and you do not need to specify an inter-engine or mediation authentication alias.

If you want to run mediations across multiple security domains, you can specify a single server identity for the bus, rather than specify a mediation authentication alias for each domain. You can also use a server identity to run mediations on the global domain.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value***. *security_value* is either `Enabled` or `Disabled`, depending on the security status of the bus.
3. Click **Launch Bus Security Wizard** to launch the wizard, or specify the following properties directly:

   Enable bus security
   Bus security is enabled by default. Clear this check box if you do not want a secure bus. The check box is read-only if administrative security is disabled.

   Inter-engine authentication alias
   The name of the authentication alias used to authorize communication between messaging engines on the bus. Specify an inter-engine authentication alias if the bus has a Version 6.x bus member, bus security is enabled, and you want to prevent unauthorized messaging engines from establishing a connection with the bus.

   Permitted transports
   Specify one of the following transports for the bus:

   - Any messaging transport chain defined to any bus member.

- Only messaging transport chains that are protected by an SSL chain.
- Only the transports specified in the list of permitted transports.

If you want to add and remove permitted transports, click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**.

Use the Server ID when running mediations
Check this option if you want to run mediations using the server identity, instead of using a mediation authentication alias.

Mediations are deployed as applications, and run in the domain used by the application server, not the bus domain. If you want to run a mediation on multiple servers in different domains, you must ensure that the user identity in the mediation authentication alias exists in the configuration for each domain. Alternatively, you can choose to use the server identity option. You can use this option when multiple domains are not in use.

Bus security domain
Specify one of the following security domains for the bus:

Global domain
You must specify the global domain if the bus contains a Version 6.x bus member, or you do not want the bus to use multiple domains.

Cell level domain
Specify the cell-level security domain if the bus has Version 7.0 bus members only, and you want the bus to share security settings with the administrative cell.

Custom domain
Specify a custom security domain if the bus has Version 7.0 bus members only, and you want the bus to use a security domain that is used by another resource, or you want to create a new security configuration for this bus.

4. Save your changes to the master configuration.

Results

You have configured security properties for the selected bus.

What to do next

You can use the administrative console to control access to the bus.

Configuring a bus to run mediations in a multiple security domain environment

Use this task to configure a secured bus so that it can run mediations successfully on bus members in different security domains.

Before you begin

The secured bus must be configured to use a non-global security domain. For more information about securing buses using multiple security domains, refer to “Securing buses” on page 1272.

About this task

If your bus topology has bus members in different security domains, you can configure the bus to allow mediations to run under the server identity. This means that a mediation can run on any server in any domain. You do not have to add a dedicated user ID for each mediation to the user repository, or maintain a mediation authentication alias.

Use the administrative console to configure a secured bus to run mediations successfully as follows:

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The security settings for the selected bus are displayed.
2. Check the option **Use the Server ID when running mediations**.
3. Click **Apply**.
4. Save your changes to the master configuration.

Results

You have configured the bus to run mediations successfully across servers in multiple security domains.

What to do next

You can use the administrative console to control access to the bus by administering users and groups in the bus connector role.

Enabling client SSL authentication

You can configure a service integration bus to allow connecting client JMS applications to authenticate using Secure Sockets Layer (SSL) certificates.

About this task

This is the parent task for the steps required to establish client SSL authentication for connections between messaging engines and JMS applications running in a client container. You need to configure the bus to allow client SSL authentication, and configure the JMS client application to perform client SSL authentication. See the following topics:

- "Configuring a bus to allow client SSL authentication."
- "Configuring JMS client applications to perform client SSL authentication" on page 1280.

Configuring a bus to allow client SSL authentication

This task describes how to configure a service integration bus to enable connecting client JMS applications to authenticate using Secure Sockets Layer (SSL) certificates.

Before you begin

You must ensure that the following tasks have been completed:

- Administrative security is enabled. For more information, see Enabling security.
- A standalone Lightweight Directory Access Protocol (LDAP) user registry has been configured for storing user and group IDs. To access the user registry, you must know a valid user ID that has the administrative role, and password, the server host and port of the registry server, and the base distinguished name (DN). For more information, see Configuring Lightweight Directory Access Protocol user registries.
- Bus security is enabled. For more information, see "Disabling bus security" on page 1273.
- JMS client applications have been configured to authenticate using client SSL certificates.

About this task

If you want to allow connecting JMS application clients to authenticate to the bus using client SSL certificates you need to define an SSL configuration. There are two parts to this task. First you use the administrative console to map SSL certificates to entries in the LDAP user registry. Secondly, you create a unique SSL configuration for each endpoint address for which you want to use client SSL authentication. Do not use the default SSL configuration for the bus.

1. Use the administrative console to define certificate filters to map an SSL certificate to an entry in the LDAP server. For more information, see Creating a Secure Sockets Layer configuration. The client SSL certificate is mapped to a user ID in the user registry.
2. Create a separate SSL configuration file for each endpoint address for server, bus member or cluster on the bus, and select that client authentication is required. For more information, see Creating a Secure Sockets Layer configuration

Results

The bus is configured to allow client SSL authentication.

What to do next

Connecting JMS client applications can now authenticate to the bus using client SSL certificates.

Configuring JMS client applications to perform client SSL authentication

This task describes how to configure JMS client applications to authenticate to the bus using client Secure Sockets Layer (SSL) authentication.

Before you begin

- You have already obtained a Secure Sockets Layer (SSL) certificate for the JMS client application.
- The JMS client application is already configured to use SSL. For more information, see SSL client properties file

About this task

This task has two objectives. First, you install the SSL certificate for the client application in the key store for the application client. Secondly, you modify the sib.client.ssl.properties file to use client SSL authentication. You use the Key Management (iKeyman) utility to work with SSL certificates. The iKeyman user interface is Java-based and uses the Java support that is installed with IBM HTTP Server.

Take the following steps to configure a JMS client application to use client SSL authentication:

1. Start the iKeyman user interface. Refer to the *iKeyman User's Guide* available from IBM developer kits for more information about using iKeyman.
2. When prompted, select the key store for the JMS client application.
3. When prompted for the type off certificate to work with, select the option **Personal certificates**. A list of personal certificates is displayed.
4. Select that you want to import a certificate to the selected key store.
5. When prompted, type the location and name for the certificate. You can provide an alias for the certificate. The certificate is installed into the keystore of the client application.
6. Close the iKeyman user interface.
7. Open a text editor to work with the sib.client.ssl.properties properties file. This file is located in the profile_root/properties directory of the application server installation, where profile_root is the directory in which profile-specific information is stored.
8. Set the value for the property com.ibm.ssl.client.clientAuthentication to *True*.
9. Set the value for the property com.ibm.ssl.client.keyStoreClientAlias to the alias name for the certificate in the client key store.
10. Save the sib.client.ssl.properties properties file.

Results

You have now configured a JMS client application to use client SSL authentication.

Adding unique names to the bus authorization policy

How to update the authorization policy for the service integration bus with unique name entries.

About this task

You should carry out this task if you are migrating from WebSphere Application Server Version 6.x to WebSphere Application Server Version 7.0. In this task, you manually run the populateUniqueNames command to query the user repository for a selected bus for unique names, and add them to the authorization policy. If you do not manually run this command, the messaging engine performs the query, and adds the missing unique names to the authorizations policy, which adversely affects the start up time.

When you migrate from a Version 6.x node to a Version 7.0 node, the authorization policy only contains the user and group security names; it does not contain the names in the user registry that uniquely define each user and group. If an LDAP user registry is in use, the unique name is the distinguished name (DN). By default, only missing unique names are added to the authorization policy. If you set the **-force** parameter, all unique name entries added to the authorization policy

1. Launch a scripting command. For more information, refer to Starting the wsadmin scripting client.
2. At the wsadmin command prompt, type the populateUniquenames command. The following example syntax queries the user repository for the unique names that match the security names for a bus called Bus 1, and adds the missing unique names to the authorization policy .

   ```
   AdminTask.populateUniquenames('[-bus Bus1]')
   ```
3. Save your changes to the master configuration repository. The following example presents the syntax:

   ```
   AdminConfig.save()
   ```

Results

The authorization policy for the bus is updated with the missing unique names.

Example

The following example updates all the unique name entries in the authorization policy for a bus called Bus 1.

```
AdminTask.populateUniqueNames(AdminTask.populateUniquenames('[-bus Bus1 -force TRUE]')
```

What to do next

Use the administrative console to administer bus security authorizations.

Administering authorization permissions

Service integration messaging security uses role-based authorization. When a user is assigned to a role, the user is granted all of the permissions that the role contains. By administering authorization permissions, you can control user access to a bus and its resources when messaging security is enabled.

Before you begin

For guidance on security authorization for a service integration bus, refer to Planning your security requirements.

About this task

When a bus is created, a set of default authorization roles is created. Default roles provide authenticated users who have the bus connector role with full access to all local destinations on the bus. By default, only members of the Server group have the bus connector role. If a specific user needs to connect to the bus, you must explicitly add that user to the bus connector role.

You can make changes to authorization permissions when messaging security is enabled or disabled. Any changes that you make when security is disabled do not have any effect until security is enabled, as described in “Disabling bus security” on page 1273.

Note: When you specify the group authorization permissions, the group distinguished name (DN) must be used. If you specify a common name (CN) for the group name, users in that group do not have the specified authorities. For more details see Standalone Lightweight Directory Access Protocol registries.

When security is enabled, by default users cannot connect to a foreign bus. If a specific user needs to connect to a foreign bus, you must explicitly add that user to the foreign bus access list.

Use the following tasks to administer authorization permissions for a bus to meet your security requirements.
- “Administering the bus connector role”
- “Administering default roles” on page 1285
- “Administering destination roles” on page 1241
- “Administering foreign bus roles” on page 1292
- “Administering topic space root roles” on page 1299
- “Administering topic roles” on page 1302
- Listing security roles for service integration by using the wsadmin tool
- Removing users and groups by using the wsadmin tool
- Removing authorization data by using the wsadmin tool

Administering the bus connector role

Service integration bus security uses role-based authorization. By administering groups of users in the bus connector role, you can control access to the local service integration bus and its resources when messaging security is enabled.

About this task

Adding a group of users to the bus connector role for a local bus grants the members of the group permission to access local bus destinations. The users and groups you want to work with must exist in the user repository.

Use the following tasks to list, add and remove groups of users in the bus connector roles using the administrative console.
- “Listing users and groups in the bus connector role”
- “Adding users and groups in the bus connector role” on page 1283
- “Removing users and groups from the bus connector role” on page 1284

Listing users and groups in the bus connector role:

Service integration bus security uses role-based authorization. By listing the users and groups in the bus connector role, you can find out which users and group members are authorized to connect to a selected secured local bus, and its resources.

Before you begin

Ensure that security is enabled for the bus. For more information, refer to “Securing buses” on page 1272.

About this task

In this task you use the administrative console to list the users and groups in the bus connector role for a selected local bus. By default, the list is empty for a new bus.

1. Log into the administrative console.

2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**.

Results

A list of the users and groups in the bus connector role for the selected bus is displayed. The list is empty for a new bus.

What to do next

You can add and delete users and groups in the bus connector role for the selected bus.

Adding users and groups in the bus connector role:

Service integration bus security uses role-based authorization. By adding users and groups to the bus connector role for a secured bus, you can control which users and group members have access to the bus and its resources.

Before you begin

- Ensure that security is enabled for the bus. For more information, refer to “Securing buses” on page 1272.
- The users and groups that you want to add to the bus connector role must exist already in the user repository.

About this task

Adding users and groups to the bus connector role enables them to connect to the bus to carry out messaging operations. You can add a user directly to the bus connector role, or indirectly by adding a group to which the user belongs. You can also add special groups of users. There are three special groups:

Server
The server identity is a WebSphere Application Server . You cannot specify the Server group for a JMS message-driven bean (MDB).

All Authenticated
This group comprises all user identities that authenticate successfully to the bus.

Everyone
The user identities in this group are anonymous, and connect to the bus without security authentication.

Note:

- If the user registry is an LDAP registry, you must use the group distinguished name (DN) when you specify a group name to add to a bus connector role. Using the common name (CN) causes problems in security authorization. For more information, refer to Service integration bus security: Troubleshooting tips and Standalone Lightweight Directory Access Protocol registries.
- If you attempt to add a user or a group that is already a member of the bus connector role, a warning message is displayed.

In this task you use an administrative console wizard to add groups and users to the bus connector role for a selected local bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**. A list of the users and groups already in the bus connector role for the selected bus is displayed. By default, the list is empty for a new bus.

3. Click **New** to launch the Security Resource Wizard.
4. Choose whether you want to add groups or users:
   - If you want to add a special group, select **The built-in special groups** option.
   - If you want to add other groups or users in the user repository, select the appropriate option, and complete the following mandatory fields:

     Search pattern
     Specify a string to match against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. You can specify wildcard characters.

     Maximum number of search results to display
     Specify the maximum number of user IDs or group names to display.
5. Click **Next** to display a list of groups or users.
6. Select the names of the groups or users you want to add to the bus connector role, and click **Next**.
7. Click **Finish** to confirm you choices.
8. Save your changes to the master configuration.

Results

The selected users and groups are added to the bus connector role for the selected bus.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from the bus connector role:

Service integration bus security uses role-based authorization. By removing selected users and groups from the bus connector role for a selected secured bus you prevent those users and group members from connecting to the bus.

About this task

The users and groups that you remove from the bus connector role for a bus can no longer perform messaging operations on the bus. Note that removing a user from the bus connector role does not prevent that user from connecting to the bus if they are also a member of a group that is in the bus connector role.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Users and groups in the bus connector role**. A list of the users and groups in the bus connector role for the selected bus is displayed.
3. Select the check box next to the name of the user or group that you want to remove, and click **Delete**.
4. Save your changes to the master configuration.

Results

The selected users and groups are removed from the bus connector role.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering default roles

Service integration bus security uses role-based authorization. By adding a user or a group to the default roles for a secured bus, you can control which users and group members have access to the bus and its resources in the default roles when messaging security is enabled.

About this task

The default roles are sender, receiver, browser, and creator. These roles apply to bus destinations that do not have a destination role, or have been configured to inherit the default roles. By default, all destinations inherit the default roles. Access in the default roles exists in addition to any specific access roles that have been configured for a destination.

Use the following tasks to list, add, and remove groups of users in default roles using the administrative console.

- "Adding users and groups to default roles"
- "Removing users and groups from default roles" on page 1286
- "Listing users and groups in default roles"

Listing users and groups in default roles:

Service integration bus security uses role-based authorization. By listing the users and groups in the default roles for a selected secured bus, you can find out which users and group members are authorized to perform messaging operations on a local bus destinations that is allowed to inherit default roles.

About this task

In this task you use the administrative console to list users and groups in the default access roles for a selected secured bus. The default role types are sender, receiver, browser and creator.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed. The information for the default access is displayed in a collapsed section.
3. Expand the Default access header.

Results

A list of users and groups in default roles for the selected bus is displayed.

What to do next

You can also add and remove users and groups in default roles.

Adding users and groups to default roles:

Service integration bus security uses role-based authorization. By adding selected users and groups to the default roles for all the local bus destinations on a secured bus, you provide those users and group members with access to the local bus destinations that are allowed to inherit default roles.

Before you begin

If a bus destination is not allowed to inherit the default roles, you must first add the user or group to the role that grants authorization permission for the specific local destination. For more information, see "Adding users and groups to destination roles" on page 1241.

About this task

The default roles are sender, receiver, creator and browser. In this task you use an administrative console wizard, the Security wizard, to add selected users or groups to the default roles. The Security wizard requests information to enable it to retrieve selected users or groups from the potentially very large number of users and groups in the user repository.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed.
3. Expand the Default access header to list the users and groups that have been assigned to default access roles.
4. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to default access roles:
   a. Search for the users or groups that you want to add to default access roles:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to the default access roles, and click **Next**. A list of user IDs or group names that you can add to the default access roles is displayed. Note that some users or groups may already be assigned to default access roles.
   d. Select the role types that you want to assign to a user or group. For example, to assign a group to the sender role, click the sender icon for the appropriate group name. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   e. Complete the previous step for each user or group that you want to add to access roles, and then click **Next**. A summary of your role type assignments is displayed.
   f. Click **Previous** to review and change your assignments, if required.
   g. Click **Finish** to confirm your assignments. The Default access roles panel is redisplayed and shows the new role type assignments.
5. Save your changes to the master configuration.

Results

The selected users and groups are added to selected default roles for the selected bus.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from default roles:

Service integration bus security uses role-based authorization. By removing selected users and groups from the default roles for a selected secured bus, you prevent those users and group members from accessing the bus using the default roles.

About this task

When you remove users and groups from the default roles for a bus, they can no longer access local bus destinations that inherit default access permissions. Note that removing a user from the default roles does not prevent that user from accessing the bus if they are also a member of a group that has been assigned to the default roles for that bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage default access roles**. The Default access roles panel is displayed.
3. Expand the Default access header to list the users and groups that have been assigned to default access roles for the selected bus.
4. Select the users and groups that you want to remove from the default access roles for this bus, and click **Remove**.
5. Save your changes to the master configuration.

Results

The selected users and groups are removed from the default roles for the selected bus. The Default access roles panel displays the changes to the default access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering destination roles

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups must have authority to perform messaging operations, at a bus destination. By administering destination roles, you can control which users and groups can perform operations at a bus destination, and the type of operation that they can perform.

About this task

You use the administrative console to administer users and groups in access roles for a destination. The access roles available for a destination depend on the type of destination. The table below lists the roles that you can assign for each destination type:

Destination type	Access roles
queue	sender, receiver, browser, creator
port	sender, receiver, browser, creator
webService	sender, receiver, browser, creator
topicSpace	sender, receiver
foreignDestination	sender
alias	sender, receiver, browser

In addition to controlling which users and groups have access to a specific local or foreign destination, you can also control the inheritance of access roles for a specific local destination. In this case, the default access roles that apply to all the destinations in the local bus namespace are added to any access roles that have been added for a specific destination.

Use the following tasks to administer destination roles.
- Listing users and groups in bus destination roles by using the wsadmin tool
- Adding users and groups to bus destination roles by using the wsadmin tool
- Removing users and groups from bus destination roles by using the wsadmin tool
- Defining destination defaults inheritance by using the wsadmin tool
- Determining destination defaults inheritance by using the wsadmin tool

Listing users and groups in destination roles:

Service integration bus security uses role-based authorization. By listing the users and groups in the destination roles for a selected secured bus, you can find out which users and groups are authorized to access the bus, and its resources.

About this task

In this task you use the administrative console to list all the users and groups in destination roles for selected destinations. The list includes users and groups that have references in the service integration role-based configuration; it does not include all the users and groups that exist in the user repository. The permitted destination roles are sender, receiver, browser and creator, depending on the destination type. Icons are used in the administrative console to represent the roles to which users and groups have been assigned. For example, if the role type set icon () is displayed in the sender role for a group called Group 1, it means that Group 1 has been assigned to the sender role for a selected destination. For a complete description of all the icons used to represent role assignments in the administrative console, refer to Access role assignments for bus security resources.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. The Destinations panel lists all the destinations defined for the selected bus.
3. Select one or more destinations to work with:
   - Click the name of a single destination.
   - Select the check boxes next to multiple destinations, and click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each selected destination is displayed in a collapsed section.
4. Expand a destination header.

Results

The Destination access roles panel lists the users and groups in access roles for the expanded destination.

What to do next

You can now administer the users and groups in destination roles at this destination.

Adding users and groups to destination roles:

Service integration bus security uses role-based authorization. By adding users and groups to the destination roles for a secured bus, you can control which users and group members can perform messaging operations at a bus destination.

Before you begin

Ensure that the following conditions are met:
- Security is enabled for the bus. For more information, refer to “Securing buses” on page 1272.

- The users and groups that you want to add to destination roles must exist already in the user repository.

About this task

By adding users or groups to destination role, you grants the users or groups authority to perform the operation defined by the role at a selected destination. The destination roles are sender, receiver, browser, and creator, depending on the destination type.

In this task you use the administrative console Security wizard to retrieve selected users or groups from the user repository, and add them to destination roles for selected bus destinations.

Note: To add a large number of users to destination roles, it is advisable to create a group in the user repository, and add the group to the destination roles.

1. Start the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles**. A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destination to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles for this destination. You can verify that the user or group you want to add does not already have a role at this destination.
5. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to access roles for the expanded destination:
   a. Search for the users or groups that you want to add to access roles for the expanded destination:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to access roles for the currently expanded destination, and click **Next**. A list of user IDs or group names that you can add to destination roles is displayed. Note that some users or groups may already be assigned to access roles for this destination.
   d. Select the appropriate access role icon for the user ID or group name that you want to add to the role at this destination. For example, select the **Receiver** icon for a user ID or group name that you want to add to the receiver role. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   e. Repeat the previous step to add more users or groups to access roles for the currently expanded destination, and then click **Next**. A summary of your access role assignments is displayed.

f. Click **Previous** to review and change your assignments, if required.
g. Click **Finish** to confirm your assignments.

6. Repeat steps 4 and 5 for each destination you want to work with.
7. Save your changes to the master configuration.

Results

The selected users and groups are added to the access roles for the currently expanded destination. The new access role assignments are displayed in the Destination access roles panel.

Example

A group called MyGroup receives messages from three queues, Queue 1, Queue 2, and Queue 3. If you want the group MyGroup to produce and consume messages at an additional destination, Queue 4, you add MyGroup to Queue 4, and then add MyGroup to the sender and receiver roles for Queue 4.

What to do next

Use the administrative console to perform other security administrative tasks.

Removing users and groups from destination roles:

Service integration bus security uses role-based authorization. By removing users and groups from the destination roles for a secured bus, you can prevent those users and group members from performing messaging operations on the bus.

About this task

When selected users and groups no longer require access to a destination, you can remove them from all the roles for that destination.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage destination access roles** A list of the destinations defined for the selected bus is displayed in the Destinations panel.
3. Select one or more destinations to work with:
   - Click a single destination name.
   - Select the check boxes next to multiple destination names, and then click **Manage Access Roles**.

   The Destination access roles panel is displayed. The information for each destination you have selected is displayed in a collapsed section.
4. Expand a destination header to list the users and groups that have been assigned to roles at this destination, and verify that the user or group that you want to remove has a role at this destination.
5. Select the users and groups that you want to remove from all role types at this destination, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from all role types at the selected destination. The Manage access roles for users and groups panel displays the updated role type assignments.

Example

The members of three groups, Group A, Group B, and Group C, belong to the sender role and the receiver role for two queue destination, Queue 1 and Queue 2. If Group B is no longer required to send and receive messages on Queue 2, you can use this task to remove Group B from all the role types on Queue 2.

What to do next

You can perform other security administration tasks by using the administrative console.

Enabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, and users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If topic role inheritance has been disabled for a particular topic, you can restore it by using the administrative console.

Before you begin

You must ensure that the following conditions are met:

- Messaging security is enabled. For more information, refer to “Disabling bus security” on page 1273.
- Topic level security is enabled for the topic space. Check the setting **Topic Access Check Required?** in the topic space destination configuration. For more information, refer to “Configuring bus destination properties” on page 1230.

About this task

In this task you use the administrative console to restore topic role inheritance for selected topics. A topic can only inherit the sender and receiver roles from the parent topic in the topic hierarchy.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.
3. Expand the topic name header to display details of the users and groups that have one or more access roles for this topic.
4. Select the **Inherit sender role from parent topic** check box.
5. Select the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The select topic inherits access roles from the parent topic. The Topic access roles panel displays the inherited access roles for the topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Disabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If you do not want topics to inherit topic roles from the parent topic in the topic hierarchy, you can override topic role inheritance by using the administrative console.

Before you begin

About this task

In this task you use the administrative console to prevent a selected topic from inheriting authorization roles from its parent topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.
3. Expand the topic name header to display details of the users and groups that have access one or more access roles for the selected topic.
4. Clear the **Inherit sender role from parent topic** check box.
5. Clear the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The selected topic cannot inherit access roles from its parent topic. The Topic access roles panel displays the changed access roles for the selected topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering foreign bus roles

Service integration bus security uses role-based authorization. When messaging security is enabled, groups of users require authority to send messages from a local bus destination to a foreign bus. By listing, adding and removing users and groups in foreign bus roles, you can control who can send messages to foreign buses.

Before you begin

These tasks assumes that one or more foreign bus connections have been configured. For more information, refer to “Configuring foreign bus connections” on page 1155.

About this task

The foreign bus connection represents another service integration bus. The bus is either in the same cell as the local bus, or in a different cell, or it represents a WebSphere® MQ queue manager. From the local bus, every other bus is regarded as a foreign bus, even if it is a bus in the same cell. Messages route to a foreign bus either directly through a link between the local bus and the foreign bus, or indirectly through one or more intermediate buses. A member of a group that belongs to the sender role on the local bus and the foreign bus can send messages directly to the foreign bus. The sender role is the only foreign bus role.

For more information about how to list, add and remove groups of users in foreign bus roles using the administrative console, refer to the following tasks.
- "Adding users and groups to foreign bus roles"
- "Removing users and groups from foreign bus roles" on page 1295
- "Listing users and groups in foreign bus roles"

Listing users and groups in foreign bus roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send messages from a secured local bus destination to a secured foreign bus. By listing all the users and groups in foreign bus roles for a selected foreign bus, you can find out who has authority to send messages from the local bus to the selected foreign bus.

About this task

In this task you use the administrative console to list users and groups in the sender role for selected foreign buses. The list includes users and groups that have references in the service integration role-based configuration. It does not include all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of all the foreign buses defined for the selected bus is displayed.
3. Select one or more foreign buses:
   - Click the name of a single foreign bus.
   - Select the check boxes next to multiple foreign buses and click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each foreign bus is displayed in a collapsible section.
4. Expand the section header for a foreign bus.

Results

A list of all the users and groups that have been assigned to the sender role for the currently expanded foreign bus is displayed.

What to do next

You can add and remove users and groups in the sender role for the selected foreign bus.

Adding users and groups to foreign bus roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send messages from a secured local bus destination to a secured foreign bus. By adding selected users and groups to the sender role for a selected foreign bus, you can control who has authority to send messages to the selected foreign bus.

Before you begin

This task assumes that the following conditions have been met:
- One or more foreign bus connections have been configured for the local bus. For more information, refer to "Configuring foreign bus connections" on page 1155.
- The users and groups that you want to add to foreign bus roles must exist in the user repository.

About this task

By default, when security is enabled, users and groups cannot send messages to a foreign bus. You need to add them to the sender role for the foreign bus. In this task you uses an administrative console wizard to select one or more foreign buses, retrieve selected users or groups from the potentially very large number of users and groups in the user repository, and add them to the sender role for the selected foreign buses.

1. Start the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of the foreign buses defined for the selected bus is displayed in the Foreign buses panel.
3. Select one or more foreign buses to work with:
   - Click a single foreign bus name.
   - Select the check boxes next to multiple foreign bus names, and then click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each foreign bus you have selected is displayed in a collapsed section.
4. Expand a foreign bus header to list the users and groups that have been assigned to roles for this foreign bus. You can verify that the user or group you want to add does not already have a role for this foreign bus.
5. Click **Add** to launch the Security wizard. The wizard takes you through the following steps to add selected users or groups to the sender role for the selected foreign bus:
   a. Search for the users or groups that you want to add to the sender role for the expanded foreign bus:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wildcard characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to add to the sender role for the currently expanded foreign bus, and click **Next**. A list of users IDs or group names that you can add to the sender role is displayed. Note that some users or groups may already be assigned to the sender role for this foreign bus.
   d. Select the **Sender** icon for a user ID or group name that you want to add to the sender role. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   e. Repeat the previous step for each user or group you want to add to the sender role, and then click **Next**. A summary of your role assignments is displayed.
   f. Click **Previous** to review and change your assignments, if required.
   g. Click **Finish** to confirm your assignments.
6. Save your changes to the master configuration.

Results

The selected users and groups are added to the sender role for the selected foreign bus. The new access roles are displayed in the Foreign bus access roles panel.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from foreign bus roles:

Service integration bus security uses role-based authorization. By removing users and groups from the foreign bus roles for a selected secured local bus, you prevent those users and groups from sending messages to the foreign bus.

About this task

In this task you use the administrative console to remove selected users or groups from the sender role for a selected foreign bus.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage foreign bus access roles**. A list of all the foreign buses defined for the selected bus is displayed.
3. Select one or more foreign buses to work with:
   - Click a single foreign bus name.
   - Select the check boxes next to multiple foreign bus names, and then click **Manage Access Roles**.

   The Foreign bus access roles panel is displayed. The access roles information for each selected foreign bus is displayed in a collapsed section.
4. Expand the section header for a foreign bus to display details for all the users and groups that have the sender role for this foreign bus.
5. Select the users and groups that you want to remove from the sender role, and click **Remove**. The selected users and groups are removed from the sender role for this foreign bus.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from the sender role for the selected foreign bus. The Foreign bus access roles panel displays the changed access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering temporary destination prefix roles

Service integration bus security uses role-based authorization. A temporary destination prefix can have two role types: creator and sender. The messaging engine uses the temporary destination prefix at runtime to determine which users and groups have authority to create a temporary destination, and send messages to temporary destinations. By administering temporary destination prefix roles for a bus, you control which users and groups can create and send messages to temporary destinations for a selected bus.

Before you begin

Ensure that security is enabled for the bus. For more information, refer to “Securing buses” on page 1272.

About this task

By default, a bus does not contain any temporary destination prefixes. You use the administrative console to add a new temporary destination prefix to the bus, and then assign selected users and groups to the sender role for the new temporary destination prefix. The creator role is assigned by default. All authenticated users can create temporary destinations by default. You can remove selected users and groups from the sender role for a selected temporary destination prefix, and you can remove a selected temporary destination prefix.

Use the following tasks to administer temporary destination prefix roles.
- "Adding users and groups to temporary destination prefix roles"
- "Removing users and groups from temporary destination prefix roles" on page 1298
- "Listing users and groups in temporary destination prefix roles"
- "Removing a temporary destination prefix" on page 1298

Listing users and groups in temporary destination prefix roles:

Service integration bus security uses role-based authorization. Temporary destination prefix roles are used to authorize access to the temporary destinations for a bus. By listing the users and groups in the temporary destination prefix roles for a selected bus, you can find out which users and groups can create temporary destinations, and send messages to temporary destinations for a selected bus.

About this task

In this task you use the administrative console to list users and groups in temporary destination prefix roles for the selected bus. The list includes users and groups that have references in the role-based security configuration for service integration. It does not include all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles**. The Temporary destination prefixes panel lists all the temporary destination prefixes defined on the bus.
3. Select one or more temporary destination prefixes to work with:
   - Click the name of a single temporary destination prefix.
   - Select the check boxes next to multiple temporary destination prefixes, and click **Manage Access Roles**.

   The Temporary destination prefix access roles panel is displayed. The access roles information for each temporary destination prefix is displayed in a collapsed section.
4. Expand the header for a selected temporary destination prefix to show its access roles.

Results

The expanded section lists the users, groups and group members that are assigned to access roles for the selected temporary destination prefix.

What to do next

You can now administer the users and groups in the sender role for a selected temporary destination prefix.

Adding users and groups to temporary destination prefix roles:

Service integration bus security uses role-based authorization. The messaging engine uses the temporary destination prefix at runtime to determine whether a client application is authorize to create, or send

messages to a particular temporary destination. By adding users and groups to temporary destination prefix roles for a selected bus, you can control which users and groups can create temporary destinations, and send messages to them.

Before you begin

The users and groups that you want to add to temporary destination prefix roles must already exist in the user repository.

About this task

By default, the bus security configuration does not contain any temporary destination prefixes. In this task, you use the administrative console Security wizard to first add a new temporary destination prefix, and then add users and groups to the sender role for the new temporary destination prefix. Note that the creator role is assigned by default to the creator of the temporary destination; you cannot use the administrative console to add users and groups to the creator role. By default, members of the All Authenticated group have authority in the creator role for temporary destination prefixes.

1. Log into the administrative console. The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus. By default, this list is empty.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles**
3. Click **Add** to launch the Security wizard:
   a. Define the name of the temporary destination prefix, and identify the users or groups that you want to add to the sender role for the temporary destination prefix:

      Resource
      This field is mandatory. Specify a name for the new temporary destination prefix.

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user identities or group names in the user repository. Only user identities or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wild card characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user identities or group names you want the administrative console to display.

   b. Click **Next**. The wizard displays the users or groups in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes for the user identities or group names that you want to assign to the sender role for the temporary destination prefix, and click **Next**. Note that you cannot assign users and groups to the creator role; it is assigned by default.
   d. Select the **Sender** icon for each user identity or group name that you want to add to the sender role. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   e. Click **Next**. A summary of your role type assignments is displayed.
   f. Click **Previous** to review and change your role type assignments. Make your changes on the Select role types page, and then click **Next**. Note that you cannot change the name of the temporary destination prefix.
   g. Click **Finish** to confirm your assignments. The role type assignments are saved to the master configuration, and the new assignments are displayed in the Temporary destination prefixes panel.

4. Save your changes to the master configuration.

Results

The selected users, groups, and group members are added to the sender role for the selected temporary destination prefix roles. The Manage access roles panel displays the new access roles.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from temporary destination prefix roles:

Service integration bus security uses role-based authorization. When security is enabled, a temporary destination prefix role is used to authorize access to temporary destinations. The temporary destination prefix is used at runtime to create temporary destinations on the bus. By removing users and groups from temporary destination prefix roles for a selected bus, you can prevent selected users and groups from sending messages to temporary destinations on the bus.

About this task

In this task you use the administrative console to remove users, groups, and group members from the sender role for selected temporary destination prefixes. Note that you cannot use this task to remove users and groups from the creator role. If you want to remove the creator role from a user or group, refer to "Removing a temporary destination prefix."

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles** The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus.
3. Select one or more temporary destination prefixes to work with:
   - Click the name of a single temporary destination prefixes.
   - Select the check boxes next to multiple temporary destination prefixes, and click Manage Access Roles.

   The Temporary destination prefix access roles panel is displayed. The access roles information for each temporary destination prefix is displayed in a collapsed section.
4. Expand the header for a selected resource to show its role type assignments.
5. Select the users and groups that you want to remove from the sender role for the currently selected temporary destination prefix, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users, groups, and group members are removed from the sender role for the selected temporary destination prefix. The Temporary destination prefix access roles panel is updated to show the changes to the access role assignments.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing a temporary destination prefix:

A temporary destination prefix is used by the service integration bus security model to determine what operations the creator of the temporary destination can perform. By removing a temporary destination prefix, you remove the creator role from the identity of the user that created the temporary destination.

About this task

A temporary destination prefix can have two authorization role types: creator and sender. In this task you use the administrative console to remove a selected temporary destination prefix from a selected bus, which removes the creator role from the user identity that created the temporary destination.

If you want to remove users and groups from the sender role for a temporary destination prefix, refer to "Removing users and groups from temporary destination prefix roles" on page 1298.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage temporary destination prefix access roles** The Temporary destination prefixes panel lists all the temporary destination prefixes defined for the selected bus.
3. Select the temporary destination prefix you want to remove.
4. Click **Remove**.
5. Save your changes to the master configuration.

Results

The selected temporary destination prefix is removed, and the creator role is removed from the user identity that created the temporary destination.

What to do next

You can perform other security administration tasks by using the administrative console.

Administering topic space root roles

Service integration bus security uses role-based authorization. When messaging security is enabled, groups of users require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By adding and removing users and groups in topic space root roles, you can control access to the topic space root.

About this task

Topic space root (/) is also called the virtual root, and it is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space, and it is a type of destination. Note that these tasks apply only to the topic space root; they do not apply to topics or a topic space. For information about administering topic access roles, refer to "Administering topic roles" on page 1302, and for information about administering topic space access roles, refer to "Administering destination roles" on page 1241.

You can add and remove users and groups in the sender and receiver roles for the topic space root. The topic space root can also inherit access in the sender and receiver roles from the topic space, providing the topic space is configured to inherit the default destination roles. For more information about topic inheritance, refer to Topic security.

For the topic space root roles to have an effect, the **Topic Access Check Required** checkbox must be set in the topic space configuration. For more information, refer to "Configuring bus destination properties" on page 1230.

Use the following tasks to list, add and remove users and groups in the topic space root roles using the administrative console.

- "Listing users and groups in topic space root roles" on page 1300

- “Adding users and groups to topic space root roles”
- “Removing users and groups from topic space root roles” on page 1302

Listing users and groups in topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By listing the users and groups in topic space root roles, you can find out who has access to the topic space root for a selected topic space.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. In this task you use the administrative console wizard to list users and groups in topic space root roles for a selected root topic.

This task applies to the topic space root only; it does not apply to the topics within the topic space, or to the topic space. If you want to list users and groups in topic roles, refer to “Listing users and groups in topic roles” on page 1303. If you want to list users and groups in a topic space (which is a type of destination), refer to “Listing users and groups in destination roles” on page 1244.

The users and groups listed in this task have references in the service integration bus security configuration. The list does not include all users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**

Results

The Topic space root panel lists all the users and groups in topic space root roles for the selected bus.

What to do next

You can administer the role type assignments for the users and groups displayed.

Adding users and groups to topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By adding users and groups to topic space root roles, you control access to the root topic in a selected topic space.

Before you begin

- The users and groups you want to add to topic space root roles must exist in the user repository.
- Topic space root roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to “Configuring bus destination properties” on page 1230.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. Note that this task applies only to the topic space root; it does not apply to adding users and groups to topics or a topic space. For information about adding users and groups to topic access roles, refer to “Adding users and groups to topic roles” on page 1303, and for adding users and groups to topic space access roles, refer to “Adding users and groups to destination roles” on page 1241.

You can add users and groups to the sender and receiver roles for the topic space root. The topic space root can also inherit access in the sender and receiver roles from the topic space, providing the topic space is configured to inherit the default destination roles. For more information about topic inheritance, refer to Topic security.

By default, a topic space does not contain a root topic. In this task you use an administrative console wizard to add a root topic to an existing topic space, retrieve the users and groups from the user repository that you want to assign to roles on the new root topic, and add them to the root topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**. The Topic spaces panel lists the topic spaces defined on the selected bus.
3. Select the name of the topic space where you want to add a new root topic. The Topics panel displays the selected topic space in a collapsible section.
4. Click **Add** to launch the Security wizard:
   a. Identify the users or groups that you want to add to the sender and receiver roles for the new root topic:

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. You can use wildcard characters in the search string.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names that you want the administrative console to display.

   b. Click **Next**. The wizard displays the new root topic, and lists the users IDs or group names in the user repository that match the information that you provided in the previous step.
   c. Select the check boxes next to the user IDs or group names that you want to assign to roles on the new root topic.
   d. Click **Next**. The wizard displays the topic role types that you can assign for the users or groups you selected in the previous step. Role types might already have been assigned for a specific user or group.
   e. Select the role types for the selected users or groups. For example, to assign a user to the sender role, select the **Sender** icon for the appropriate user ID. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   f. Click **Next**. A summary of your role type assignments for the root topic is displayed.
   g. If you want to change your assignments, click **Previous** to return to the Select role types page, change your assignments, and then click **Next**.
   h. Click **Finish** to confirm your assignments. The role type assignments are saved to the master configuration, and the new assignments are displayed in the Topics panel.
5. Save your changes to the master configuration.

Results

The selected users and groups are added to topic space root roles for the new root topic. The **Manage access roles** panel displays the new access role assignments.

What to do next

You can perform additional security administration using the administrative console.

Removing users and groups from topic space root roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to send and receive messages from the topic space root in a publish/subscribe topic hierarchy. By removing users and groups from topic space root roles, you prevent them from accessing the root topic in a selected topic space.

About this task

Topic space root (/) is the highest level topic in a publish/subscribe topic hierarchy. The hierarchy itself is called the topic space. Note that this task applies only to the topic space root; it does not apply to removing users and groups from topics or a topic space. For information about removing users and groups from topic access roles, refer to “Removing users and groups from topic roles” on page 1304, and for removing users and groups from topic space roles, refer to “Removing users and groups from destination roles” on page 1243.

In this task you use the administrative console to remove selected users and groups from the sender and receiver roles for the selected root topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles**. The Topic spaces panel lists the topic spaces defined on the bus.
3. Select the topic space you want to work with. The selected topic space is displayed in the Topics panel. The root topic (/) is displayed by default.
4. Select the topic space root. The Topic access roles panel lists the role type assignments for the topic space root.
5. Select the names of the users, groups and group members that you want to remove from all role types for the selected root topic, and click **Remove**.
6. Save your changes to the master configuration.

Results

The selected users and groups are removed from all roles for the selected root topic. The Topic access roles panel is updated to show the changes to the access roles assignments.

What to do next

You can perform other security administration commands using the administrative console.

Administering topic roles

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to access a topic in a publish/subscribe topic hierarchy. By adding and removing users and groups in topic roles, you can control access to the topic.

About this task

You use the administrative console to list, add and remove users and groups in the sender and receiver roles, and to define topic role inheritance. By default, a child topic inherits its topic roles from its parent topic. You can change the default roles for a particular topic by adding or removing topic roles at the topic level. You can also allow or block inheritance of topic roles at topic level.

You can add access roles for a topic before it exists. Topics are created at runtime only, and exist only for as long as they are active.

For information about how to list, add, and remove users and groups in topic roles using the administrative console, refer to the following tasks.
- “Listing users and groups in topic roles”
- “Adding users and groups to topic roles”
- “Removing users and groups from topic roles” on page 1304
- “Enabling topic role inheritance” on page 1291
- “Disabling topic role inheritance” on page 1291

Listing users and groups in topic roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, users and groups require authority to access topics in a publish/subscribe topic hierarchy. By listing the users and groups that are members of topic roles for a selected topic, you can find out who has authority to send messages to and from the topic.

About this task

In this task you use the administrative console to list users and groups that have access roles for a selected topic in a selected topic space. The list includes users, groups and group members that have references in the role-based security configuration for service integration. It does not list all the users and groups that exist in the external user repository.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topics panel displays the information for the topic in a collapsed section.
3. Expand the section header to display the users and groups that are assigned to role types for the selected topic.

What to do next

You can add and remove users and groups in the topic roles for the selected topic.

Adding users and groups to topic roles:

Service integration bus security uses role-based authorization. When messaging security, and topic level authorization is enabled, users and groups must be authorized to access topics in a publish/subscribe topic hierarchy. By adding users and groups to topic roles, you control access to a topic in a selected topic space.

Before you begin

- The users and groups you want to add to topic space root roles must exist in the user repository.
- Topic roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to “Configuring bus destination properties” on page 1230.

About this task

Topics are organized into one or more hierarchies within a topic space. If the **Topic Access Check Required** setting is enabled for the topic space, a user must have authorization to access the topic itself. You can add access roles to a topic before it is created at runtime. A topic inherits access roles from its parent unless you explicitly block the inheritance. For more information, refer to “Enabling topic role inheritance” on page 1291.

In this task you use an administrative console wizard to add users or groups to the sender and receiver roles for a selected topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic space root panel lists the users and groups that are assigned to role types for the selected topic.
3. Click **Add** to launch the Security wizard:
   a. Provide the following information to enable the wizard to identify the users or groups that you want to add to role types for the selected topic:

      Resource
      Specify the name of the topic.

      Users or Groups
      Select either **Users** or **Groups** to specify whether you want to grant access roles to users or groups.

      Search pattern
      This field is mandatory. Specify a search string that is matched against user IDs or group names in the user repository. Only user IDs or group names that match the search pattern are retrieved, subject to the maximum number of search results. Wild card characters are allowed.

      Maximum number of search results to display
      This field is mandatory. Specify the maximum number of user IDs or group names you want the administrative console to display.

   b. Click **Next**. The wizard lists the users IDs or group names that match the information that you provided in the previous step.
   c. Select the check boxes for the user IDs or group names that you want to assign to roles for the selected topic.
   d. Click **Next**. The wizard lists the topic role types that you can assign for the users or groups you selected in the previous step. Role types may already have been assigned for a specific user or group.
   e. Select the role types for each of the selected users or groups. For example, to assign a user ID to the sender role, select the **Sender** icon for that user ID. The icon changes from  to  to show that you have added the user or group to the access role for the resource.
   f. Click **Next**. A summary of your role type assignments for the selected topic is displayed.
   g. If you want to change your assignments, click **Previous** to return to the Select role types step. Make changes to your assignments, and click **Next** to return to the Confirm step.
   h. Click **Finish** to confirm your assignments and save your changes to the master configuration.

Results

The updated role type assignments for the selected users or groups are displayed in the Topic access roles panel.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing users and groups from topic roles:

Service integration bus security uses role-based authorization. When messaging security is enabled, and the **Topic access check required** setting is enabled for the topic space, users and groups require

authority to access a topic in the topic space. By removing users and groups from all topic roles for a selected topic, you prevent them from accessing the topic.

Before you begin

Topic roles are effective only when the **Topic Access Check Required** setting is enabled in the configuration for a topic space. For more information, refer to “Configuring bus destination properties” on page 1230.

About this task

This task uses the administrative console to remove users and groups from both the sender and receiver roles for a selected topic in a selected topic space.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel is displayed. The information for the topic is displayed in a collapsed section.
3. Expand the section header to display the users and groups that are assigned to role types for the selected topic.
4. Select the users and groups that you want to remove from the sender and receiver roles for the selected topic, and click **Remove**.
5. Save your changes to the master configuration.

Results

The selected users and groups are removed from the sender and receiver roles for the selected topic. The Topic access roles panel is updated to show that the selected users and groups have no topic role type assignments.

Enabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, and users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If topic role inheritance has been disabled for a particular topic, you can restore it by using the administrative console.

Before you begin

You must ensure that the following conditions are met:

- Messaging security is enabled. For more information, refer to “Disabling bus security” on page 1273.
- Topic level security is enabled for the topic space. Check the setting **Topic Access Check Required?** in the topic space destination configuration. For more information, refer to “Configuring bus destination properties” on page 1230.

About this task

In this task you use the administrative console to restore topic role inheritance for selected topics. A topic can only inherit the sender and receiver roles from the parent topic in the topic hierarchy.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.

3. Expand the topic name header to display details of the users and groups that have one or more access roles for this topic.
4. Select the **Inherit sender role from parent topic** check box.
5. Select the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The select topic inherits access roles from the parent topic. The Topic access roles panel displays the inherited access roles for the topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Disabling topic role inheritance:

Service integration bus security uses role-based authorization. When messaging security, and topic level security are enabled, users and groups require access in the sender and receiver roles to access a topic in a publish/subscribe topic hierarchy. By default, topics inherit these roles from the parent topic. If you do not want topics to inherit topic roles from the parent topic in the topic hierarchy, you can override topic role inheritance by using the administrative console.

Before you begin

About this task

In this task you use the administrative console to prevent a selected topic from inheriting authorization roles from its parent topic.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage topic access roles** → ***topic_space_name*** → ***topic_name***. The Topic access roles panel lists users and groups that have been assigned role types for the selected topic.
3. Expand the topic name header to display details of the users and groups that have access one or more access roles for the selected topic.
4. Clear the **Inherit sender role from parent topic** check box.
5. Clear the **Inherit receiver role from parent topic** check box.
6. Click **OK** to save your changes.
7. Save your changes to the master configuration.

Results

The selected topic cannot inherit access roles from its parent topic. The Topic access roles panel displays the changed access roles for the selected topic.

What to do next

You can perform other security administration tasks by using the administrative console.

Removing access roles from unknown users and groups

Service integration bus security uses role-based authorization. Users and groups are assigned to access roles for specific bus resources. If a user or a group that has access roles is removed from the user repository, it becomes an unknown user. You can identify the unknown users and groups for a selected bus, and removes their access roles.

About this task

In this task, you use the administrative console to remove access roles from users and groups for unknown users and groups.

1. Click **Service integration** → **Buses** → ***security_value*** → **[Authorization Policy] Manage users and groups not known to the user repository** The Unknown users and groups panel lists all the users and groups that have access roles assigned to them, but they are not known in the user registry.
2. Select the check boxes next to multiple user and group names.
3. Click **Remove all roles**. The access role assignments for the selected users and groups are removed.
4. Save your changes to the master configuration.

Results

The access role assignments are removed from the selected users and groups. The Unknown users and groups panel is updated to show the changes to the role type assignments.

Administering permitted transports for a bus

Use these tasks to configure a transport policy for a service integration bus, and to administer the transports chains that remote applications clients can use to connect to a service integration bus.

About this task

- "Administering permitted transports for a bus"
- "Listing permitted transports for a bus" on page 1308
- "Adding a permitted transport to a bus" on page 1308
- "Removing a permitted transport from a bus" on page 1309

Configuring a transport policy for a bus

By configuring a transport policy for the bus you can affect the security of messages in transit.

Before you begin

There are no prerequisites for this task.

About this task

The transport policy for a bus controls which transport mechanism remote application clients can use to connect to the bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value***. The configuration panel for the selected bus is displayed
3. Choose one of the following transport policies for the bus:

   Allow the use of all defined transport channel chains
   Select this option to allow the bus to use unsecured ports.

   Restrict the use of defined transport channel chains to those protected by SSL
   Select this option to prevent the use of the InboundBasicMessaging port.

Restrict the use of defined transport channel chains to the list of permitted transports
Select this option if you want connecting client applications to use named transport channel chains. This provides the highest level of control over the use of transport channel chains.

4. Click **Apply**.
5. Save your changes to the master configuration.

Results

The transport policy you have configured for the selected bus controls how application clients connect to the bus.

What to do next

You can use the administrative console to add and remove transport chains in the list of permitted transports for the bus.

Listing permitted transports for a bus

Use this task to display a list of the transport chains that are available for a remote client application to use to connect to a selected service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 1307.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. In this task, you use the administrative console to list all the permitted transports for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**.

Results

The Permitted transports panel displays the list of permitted transports for the selected bus.

What to do next

You can use the administrative console to add and remove transport chains in the list of permitted transports to control which transport chains a remote client application can use to connect to the bus.

Adding a permitted transport to a bus

Use this task to make a new transport chain available to a remote client application to use to connect to a service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 1307.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. If you want to allow remote client applications to connect to the bus using a new transport chain, you must add the new transport chain to the list of permitted transports for the bus. In this task, you use the administrative console to add a new transport chain to the list of permitted transports for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**. The Permitted transports panel displays the list of permitted transports for the selected bus.
3. Click **New**.
4. Select the name of the transport chain you want to add in the list box.
5. Click **OK**.
6. Save your changes to the master configuration.

Results

The new transport name is displayed in the list of permitted transports, and its is available for use by a remote client application to connect to the bus.

What to do next

You can use the administrative console to remove transport chains from a bus.

Removing a permitted transport from a bus

Use this task to remove a selected transport chain from the list of permitted transport chains for a selected service integration bus.

Before you begin

The transport policy for the bus must be set to the option **Restrict the use of defined transport channel chains to the list of permitted transports** for this task to have an effect. For more information about how to configure the transport policy for the bus, refer to “Configuring a transport policy for a bus” on page 1307.

About this task

A permitted transport is a transport chain that a remote client application can use to connect to the bus. If you want to prevent a remote client application from using a particular transport chain to connect to the bus, you can remove the transport chain from the list of permitted transports for the bus. In this task, you use the administrative console to remove a transport chain from the list of permitted transports for a selected bus.

1. Log onto the administrative console.
2. Click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**. The Permitted transports panel displays the list of permitted transports for the selected bus.
3. Select the name of the transport chain you want to remove.
4. Click **Delete**. The Permitted transports panel displays an updated list of permitted transports for the selected bus.
5. Save your changes to the master configuration.

Results

The selected transport chain is removed from the list of permitted transports for the selected bus, and a remote client can no longer use it to connect to the bus.

What to do next

You can use the administrative console to manage the transport policy for the bus.

Configuring bus security using an administrative console panel

Use this task to enable or disable messaging security, and configure security properties for an existing service integration bus.

Before you begin

The bus must exist in the administrative console. If you want to create a new bus, refer to “Adding buses” on page 1140.

About this task

This task uses the Bus Security administrative console panel. You can launch the Bus security wizard from the panel, or specify individual security properties directly in the panel. The bus security properties are effective only when administrative security for the cell is enabled. If the wizard detects that administrative security is disabled, it prompts you to enable it.

The security properties available to a particular bus depend on the versions of the bus members. If the bus has a WebSphere Application Server Version 6.x bus member, you must specify the global security domain. The bus cannot use cell level, or custom security domains. You must also specify an inter-engine authentication alias to prevent unauthenticated messaging engines from establishing a connection with the bus. If the bus contains Version 7.0 bus members only, you can specify any type of security domain, and you do not need to specify an inter-engine or mediation authentication alias.

If you want to run mediations across multiple security domains, you can specify a single server identity for the bus, rather than specify a mediation authentication alias for each domain. You can also use a server identity to run mediations on the global domain.

1. Log into the administrative console.
2. Click **Service integration** → **Buses** → ***security_value***. *security_value* is either `Enabled` or `Disabled`, depending on the security status of the bus.
3. Click **Launch Bus Security Wizard** to launch the wizard, or specify the following properties directly:

   Enable bus security
   Bus security is enabled by default. Clear this check box if you do not want a secure bus. The check box is read-only if administrative security is disabled.

   Inter-engine authentication alias
   The name of the authentication alias used to authorize communication between messaging engines on the bus. Specify an inter-engine authentication alias if the bus has a Version 6.x bus member, bus security is enabled, and you want to prevent unauthorized messaging engines from establishing a connection with the bus.

   Permitted transports
   Specify one of the following transports for the bus:
   - Any messaging transport chain defined to any bus member.
   - Only messaging transport chains that are protected by an SSL chain.
   - Only the transports specified in the list of permitted transports.

If you want to add and remove permitted transports, click **Service integration** → **Buses** → ***security_value*** → **[Additional Properties] Permitted Transports**.

Use the Server ID when running mediations
Check this option if you want to run mediations using the server identity, instead of using a mediation authentication alias.

Mediations are deployed as applications, and run in the domain used by the application server, not the bus domain. If you want to run a mediation on multiple servers in different domains, you must ensure that the user identity in the mediation authentication alias exists in the configuration for each domain. Alternatively, you can choose to use the server identity option. You can use this option when multiple domains are not in use.

Bus security domain
Specify one of the following security domains for the bus:

Global domain
You must specify the global domain if the bus contains a Version 6.x bus member, or you do not want the bus to use multiple domains.

Cell level domain
Specify the cell-level security domain if the bus has Version 7.0 bus members only, and you want the bus to share security settings with the administrative cell.

Custom domain
Specify a custom security domain if the bus has Version 7.0 bus members only, and you want the bus to use a security domain that is used by another resource, or you want to create a new security configuration for this bus.

4. Save your changes to the master configuration.

Results

You have configured security properties for the selected bus.

What to do next

You can use the administrative console to control access to the bus.

Securing messages between messaging buses

Use these tasks to administer the access control security associated with sending messages between buses.

About this task

- "Protecting messages transmitted between buses"
- "Administering access to foreign destinations" on page 1312

Protecting messages transmitted between buses

Use this task to protect the integrity of the data that is transmitted between secured linked service integration buses.

Before you begin

- Review the information in the topics Secure transport considerations and Configuring transport chains.
- Configure a transport policy for the bus, as described in the topic "Configuring a transport policy for a bus" on page 1307.

About this task

In this task, you configure transport chains for the remote bus to ensure that only secured inbound transport chains can contact the messaging engines on the server. The remote bus may be a foreign bus, or an WebSphere MQ link.

1. Ensure that the linked buses are secured. For further information, see “Securing buses” on page 1272.
2. Configure the Target inbound transport chain property for the foreign bus, or WebSphere MQ link, as appropriate:
   - For connections to a foreign bus, refer to “Connecting service integration buses to use point-to-point messaging” on page 1165 or “Connecting service integration buses to use publish-subscribe messaging” on page 1167
   - For connections to WebSphere MQ, refer to “Creating a new WebSphere MQ link” on page 1615.

Results

You have configured the Target inbound transport chains for the remote service integration bus.

What to do next

Administering access to foreign destinations

Use these tasks to administer the access control security associated with sending messages to foreign bus destinations.

About this task

To define the authentication information used in the access control checks that are performed when a message is sent to a destination in a foreign bus, complete the following steps:

- Define the required authorization permissions to allow a sender to access a destination on a foreign bus. These can be defined using either a foreign bus definition, as described in “Administering foreign bus roles” on page 1292, or a foreign bus destination definition, as described in “Administering destination roles” on page 1241. These permissions are used when the message is sent, to check that the sender is allowed to access the foreign bus.
- The administrator of the foreign bus must define the required authorization permissions to allow the messages to access the destination on the foreign bus. These permissions are used when the message enters the foreign bus.
- Define the authorization permissions to allow any messages entering your bus from the foreign bus to access the required destinations.

Securing access to a foreign bus

You can secure the link between a local bus and a foreign bus.

Before you begin

Before you can secure the link between a local bus and a foreign bus, there must be foreign bus connection on the local bus, and therefore a link between the buses.

About this task

This task summarizes the significant tasks to secure the link between a local bus and a foreign bus. For more general information about service integration bus security, see “Security” on page 1128.

When you create a foreign bus connection, there are some options to secure the connection during that procedure.

1. Enable security on the service integration bus and the foreign bus. See “Disabling bus security” on page 1273.
2. Secure the link between the buses. See “Securing messages between messaging buses” on page 1311.
3. Grant access to the local bus for users who will be sending messages to the foreign bus. See “Administering the bus connector role” on page 1282.
4. Grant access to the foreign bus for users who will be sending messages to it. See “Administering foreign bus roles” on page 1292.
5. Optional: Give users access to foreign or alias destinations that will forward messages to a foreign bus. See “Administering destination roles” on page 1241.

Securing links between messaging engines

When security is enabled, messaging engines in a mixed version bus establish trust using an authentication alias.

Before you begin

You must ensure that the user ID that you intend to use for the authentication alias meets the following conditions:

- It exists in the user registry
- It is used only for the purpose of messaging engine to messaging engine authentication.
- It has not been added to the bus connector access role.

About this task

If you have a secure, mixed version bus, you must define an Inter-engine authentication alias for use in preventing unauthorized messaging engines from establishing a connection. The Inter-engine authentication alias is used by messaging engines to establish trust in the following scenarios:

- A WebSphere Application Server Version 6.x messaging engine initiates a link with a Version 7.0 messaging engine.
- A Version 7.0 messaging engine initiates a link with a Version 6.x messaging engine.

Trust between Version 7.0 messaging engines is established using a Lightweight Third Party Authentication (LTPA) token.

1. In the navigation pane, click **Service integration** → **Buses** → ***security_value***. The bus security configuration panel is displayed.
2. In the **Inter-engine authentication alias** field, select an authentication alias.
3. Click **OK**.
4. Save your changes to the master configuration.

Results

You have selected an Inter engine authentication alias for the bus to use in establishing trust between mixed version messaging engines.

What to do next

If you require additional security, you can configure the SSL certificate stores to restrict objects that can make an SSL connection, and thereby connect to the bus. For more information see Configuring secure sockets layer.

Controlling which foreign buses can link to your bus

Use this task to control which foreign buses are allowed to link to your bus.

About this task

When messaging security is enabled, it is important that only authorized foreign buses are allowed to link to your bus.

To control which foreign buses can create a link to your bus, complete the following steps:

1. Set the authentication alias property on the foreign bus connection to be used for authentication of the foreign bus joining your bus, as described in “Connecting service integration buses to use point-to-point messaging” on page 1165 or “Connecting service integration buses to use publish-subscribe messaging” on page 1167.
2. If you require extra security, configure the SSL certificate stores to restrict who can make an SSL connection, and hence link to the bus. For more information, see Configuring secure sockets layer.

Securing database access

How to protect the data store from access by unauthorized users.

About this task

To secure access to the data store, you must configure a user name and password on the data store for the messaging engine. If you want to apply additional levels of security such as encrypting the contents of the database, use the specific security features provided by your database.

Securing mediations

Use the following tasks to secure mediations at an operations level. For example, a mediation inherits its identity from a the messaging engine, but you might want to specify an alternative identity for the mediation to use.

Chapter 7. Data access resources

Task overview: Accessing data from applications

Various enterprise information systems (EIS) use different methods for storing data. These *backend* data stores might be relational databases, procedural transaction programs, or object-oriented databases.

About this task

IBM WebSphere Application Server provides several options for accessing an information system's backend data store:

- Programming directly to the database through the JDBC 4.0 API, JDBC 3.0 API, or JDBC 2.0 optional package API.
- Programming to the procedural backend transaction through various Jave EE Connector Architecture (JCA) 1.0 or 1.5 compliant connectors.
- Programming in the bean-managed persistence (BMP) bean or servlets indirectly accessing the backend store through either the JDBC API or JCA compliant connectors.
- Using container-managed persistence (CMP) beans.
- Using the IBM data access beans, which also use the JDBC API, but give you a rich set of features and function that hide much of the complexity associated with accessing relational databases.

For all of these options, except for using the JCA 1.0 or 1.5 compliant connectors, the prerequisite Web site details which databases and drivers are currently supported.

1. Develop data access applications. Develop your application to access data using the various ways available through the application server. You can access data through APIs, container-managed persistence beans, bean-managed persistence beans, session beans, or Web components.
2. Assemble data access applications using the assembly tool. Assemble your application by creating and mapping resource references.
3. Prepare for deployment: Ensure that the appropriate database objects are available. Create or configure any databases or tables required, set necessary configuration parameters to handle expected load, and configure any necessary JDBC providers and data source objects for servlets, enterprise beans, and client applications to use.
4. Install the application on your application server.

Resource adapters

A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

WebSphere Application Server supports JCA versions 1.0 and 1.5 including additional, configurable features for JCA 1.5 resource adapters with activation specifications that handle inbound requests.

Data access for container-managed persistence (CMP) beans is indirectly managed by the WebSphere Persistence Manager. The JCA specification supports persistence manager delegation of the data access to the JCA resource adapter without knowing the specific backend store. For the relational database access, the persistence manager uses the relational resource adapter to access the data from the database.

You can find the supported database platforms for the JDBC API at the WebSphere Application Server prerequisite Web site.

Java EE Connector Architecture resource adapters

An application server vendor extends its system once to support the Java Platform, Enteprise Edition Connector Architecture (JCA) and is then assured of seamless connectivity to multiple EISs. Likewise, an EIS vendor provides one standard resource adapter with the capability to plug into any application server that supports the connector architecture.

The product supports any resource adapter that implements version 1.0 or 1.5 of this specification. IBM supplies resource adapters for many enterprise systems separately from the WebSphere Application Server package, including (but not limited to): the Customer Information Control System (CICS®), Host On-Demand (HOD), Information Management System (IMS™), and Systems, Applications, and Products (SAP) R/3 .

The general approach to writing an application that uses a JCA resource adapter is to develop EJB session beans or services with tools such as Rational Application Developer. The session bean uses the *javax.resource.cci* interfaces to communicate with an enterprise information system through the resource adapter.

WebSphere Relational Resource Adapter

WebSphere Application Server provides the WebSphere Relational Resource Adapter implementation. This resource adapter provides data access through JDBC calls to access the database dynamically. The connection management is based on the JCA connection management architecture and provides connection pooling, transaction, and security support. The WebSphere RRA is installed and runs as part of WebSphere Application Server, and needs no further administration.

The RRA supports both the configuration and use of JDBC data sources and Java EE Connection Architecture (JCA) connection factories. The RRA supports the configuration and use of data sources implemented as either JDBC data sources or Java EE Connector Architecture connection factories. Data sources can be used directly by applications, or they can be configured for use by container-managed persistence (CMP) entity beans.

For more information about the WebSphere Relational Resource Adapter, see the following topics:

- For information about resource adapters and data access, see “Data access portability features” on page 1317
- For RRA settings, see “WebSphere relational resource adapter settings”
- For information about enterprise beans, see EJB applications

WebSphere relational resource adapter settings

Use this page to view the settings of the WebSphere relational resource adapter. This adapter is preinstalled in the product to provide access to relational databases.

Note: Although the default relational resource adapter settings are viewable, you cannot make changes to them.

To view this administrative console page, click **Resources** > **Resource adapters** > **Resource adapters**. Expand the **Preferences** section at the top of the page. Select **Show built-in resources**. The table of configured resource adapters now displays the **WebSphere Relational Resource Adapter**.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a description of the relational resource adapter.

Data type String

Archive path:

Specifies the path to the Resource Adapter Archive (RAR) file containing the module for this resource adapter.

Data type String

Class path:

Specifies a list of paths or Java Archive (JAR) file names, which together form the location for the resource provider classes.

Data type String

Native path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Data type String

Data access portability features

These interfaces work with the relational resource adapter (RRA) to make database-specific functions operable on connections between the application server and that database.

In other words, your applications can access data from different databases, and use functions that are specific to the database, without any code changes. Additionally, WebSphere Application Server enables you to plug in a data source that is not supported by WebSphere persistence. However, the data source *must* be implemented as either the *XADataSource* type or the *ConnectionPoolDataSource* type, and it must be in compliance with the JDBC 2.x specification.

You can achieve application portability through the following:

DataStoreHelper interface

With this interface, each data store platform can plug in its own private datastore specific functions that the relational resource adapter runtime uses. WebSphere Application Server provides an implementation for each supported JDBC provider.

The interface also provides a GenericDataStoreHelper class for unsupported data sources to use. You can subclass the GenericDataStoreHelper class or other WebSphere provided helpers to support any new data source.

Note: If you are configuring data access through a user-defined JDBC provider, do not implement the DataStoreHelper interface directly. Either subclass the GenericDataStoreHelper class or subclass one of the DataStoreHelper implementation classes provided by IBM (if your database behavior or SQL syntax is similar to one of these provided classes).

For more information, see the API documentation **DataStoreHelper** topic (as listed in the API documentation index).

The following code segment shows how a new data store helper is created to add new error mappings for an unsupported data source.

```
public class NewDSHelper extends GenericDataStoreHelper
{
  public NewDSHelper(java.util.Properties dataStoreHelperProperties)
  {
    super(dataStoreHelperProperties);
    java.util.Hashtable myErrorMap = null;
    myErrorMap = new java.util.Hashtable();
    myErrorMap.put(new Integer(-803), myDuplicateKeyException.class);
    myErrorMap.put(new Integer(-1015), myStaleConnectionException.class);
    myErrorMap.put("S1000", MyTableNotFoundException.class);
    setUserDefinedMap(myErrorMap);
    ...
  }
}
```

WSCallHelper class

This class provides two methods that enable you to use vendor-specific methods and classes that do not conform to the standard JDBC APIs (and are not part of WebSphere Application Server extension packages).

- **jdbcCall() method**

  By using the static jdbcCall() method, you can invoke vendor-specific, nonstandard JDBC methods on your JDBC objects. (For more information, see the API documentation **WSCallHelper** topic.) The following code segment illustrates using this method with a DB2 data source:

  ```
  Connection conn = ds.getConnection();
  // get connection attribute
  String connectionAttribute =(String) WSCallHelper.jdbcCall(DataSource.class, ds,
   "getConnectionAttribute", null, null);
  // setAutoClose to false
  WSCallHelper.jdbcCall(java.sql.Connection.class,
  conn, "setAutoClose",
  new Object[] { new Boolean(false)},
  new Class[] { boolean.class });
  // get data store helper
  DataStoreHelper dshelper = WSCallHelper.getDataStoreHelper(ds);
  ```

- **jdbcPass() method**

  Use this method to exploit the nonstandard JDBC classes that some database vendors provide. These classes contain methods that require vendors' proprietary JDBC objects to be passed as parameters.

  In particular, implementations of Oracle can involve use of nonstandard classes furnished by the vendor. Methods contained within these classes include:

  ```
  oracle.sql.ArrayDescriptor ArrayDescriptor.createDescriptor(java.lang.String, java.sql.Connection)
  oracle.sql.ARRAY new ARRAY(oracle.sql.ArrayDescriptor, java.sql.Connection, java.lang.Object)
  oracle.xml.sql.query.OracleXMLQuery(java.sql.Connection, java.lang.String)
  oracle.sql.BLOB.createTemporary(java.sql.Connection, boolean, int)
  oracle.sql.CLOB.createTemporary(java.sql.Connection, boolean, int)
  oracle.xdb.XMLType.createXML(java.sql.Connection, java.lang.String)
  ```

  The following code sample demonstrates how to use jdbcPass to call the Oracle method `XMLType.createXML` on a connection. This Oracle function creates an XML type object out of the XML data that the database passes to your application.

  ```
  XMLType poXML = (XMLType)(WSCallHelper.jdbcPass(XMLType.class,
  "createXML", new Object[]{conn,poString},
   new Class[]{java.sql.Connection.class, java.lang.String.class},
   new int[]{WSCallHelper.CONNECTION,WSCallHelper.IGNORE}));
  ```

  For more examples of using jdbcPass and a complete list of method parameters, see the API documentation for the WSCallHelper class. In this information center, access the API documentation with the following steps:

1. Click **Reference** > **Developer API documentation** > **Application programming interfaces**
2. Click **com.ibm.websphere.rsadapter**
3. Under the Class Summary heading, click **WSCallHelper**

The first section on jdbcPass discusses using the method to call database static methods. The second section on jdbcPass addresses database non-static methods.

Note: Use of the jdbcPass() method causes the JDBC object to be used outside of the protective mechanisms of WebSphere Application Server. Performing certain operations (such as setting autoCommit, or transaction isolation settings, etc.) outside of these protective mechanisms will cause problems with the future use of these pooled connections. IBM does not guarantee stability of the object after invocation of this method; it is the user's responsibility to ensure that invocation of this method does not perform operations that harm the object. Use at your own risk.

Because of these potential problems, WebSphere Application Server strictly controls which methods are allowed to be invoked using the jdbcPass() method support. If you require support for a method that is not listed previously in this document, contact WebSphere Application Server Support with information on the method you require.

JDBC providers

Installed applications use JDBC providers to interact with relational databases.

The JDBC provider object supplies the specific JDBC driver implementation class for access to a specific vendor database. To create a pool of connections to that database, you associate a data source with the JDBC provider. Together, the JDBC provider and the data source objects are functionally equivalent to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity with a non-relational database.

For a current list of supported providers, see the WebSphere Application Server prerequisite Web site. For detailed descriptions of the providers, including the supported data source classes and their required properties, refer to the topics on data source required minimum required settings, by vendor.

Data sources

Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

A data source is associated with a JDBC provider, which supplies the driver implementation classes that are required for JDBC connectivity with your specific vendor database. Application components transact directly with the data source to obtain connection instances to your database. The connection pool that corresponds to each data source provides connection management.

You can create multiple data sources with different settings, and associate them with the same JDBC provider. For example, you might use multiple data sources to access different databases within the same vendor database application. WebSphere Application Server requires JDBC providers to implement one or both of the following data source interfaces, which are defined by Sun Microsystems. These interfaces enable the application to run in a single-phase or two-phase transaction protocol.

- *ConnectionPoolDataSource* - a data source that supports application participation in local and global transactions, excepting two-phase commit transactions. When a connection pool data source is involved in a global transaction, transaction recovery is not provided by the transaction manager. The application is responsible for providing the backup recovery process if multiple resource managers are involved.

  Note: A connection pool data source does support two-phase commit transactions in these cases:

  - the data source is making use of *Last participant* support. Last participant support enables a single one-phase commit resource to participate in a global transaction with one or more two-phase commit resources.

    For more information, consult the article Using one-phase and two-phase commit resources in the same transaction.
- *XADataSource* - a data source that supports application participation in any single-phase or two-phase transaction environment. When this data source is involved in a global transaction, the product transaction manager provides transaction recovery.

Prior to version 5.0 of the application server, the function of data access was provided by a single connection manager (CM) architecture. This connection manager architecture remains available to support Java 2 Platform, Enterprise Edition (J2EE) 1.2 applications, but another connection manager architecture is provided, based on the JCA architecture supporting the J2EE 1.3 application style, J2EE 1.4 and Java EE applications.

These architectures are represented by two types of data sources. To choose the right data source, administrators must understand the nature of their applications, EJB modules, and enterprise beans.
- Data source (WebSphere Application Server V4) - This data source runs under the original CM architecture. Applications using this data source behave as if they were running in Version 4.0.
- Data source - This data source uses the JCA standard architecture to provide support for J2EE version 1.3 and 1.4, as well as Java EE applications. It runs under the JCA connection manager and the relational resource adapter.

Choice of data source
- J2EE 1.2 application - all EJB 1.1 enterprise beans, JDBC applications, or Servlet 2.2 components must use the **4.0** data source.
- J2EE 1.3 (and subsequent releases) application -
  - EJB 1.1 module - all EJB 1.x beans must use the **4.0** data source.
  - EJB 2.0 (and subsequent releases) module - enterprise beans that include container-managed persistence (CMP) Version 1.x, 2.0, and beyond must use the **new** data source.
  - JDBC applications and Servlet 2.3+ components - must use the **new** data source.

Data access beans
Data access beans provide a rich set of features and function, while hiding much of the complexity associated with accessing relational databases.

They are Java classes written to the Enterprise JavaBeans specification.

You can use the data access beans in JavaBeans-compliant tools, such as the IBM *Rational Application Developer*. Because the data access beans are also Java classes, you can use them like ordinary classes.

The data access beans (in the package *com.ibm.db*) offer the following capabilities:

Feature
: **Details**

Caching query results
: You can retrieve SQL query results all at once and place them in a cache. Programs using the result set can move forward and backward through the cache or jump directly to any result row in the cache.

  For large result sets, the data access beans provide ways to retrieve and manage *packets*, subsets of the complete result set.

Updating through result cache
: Programs can use standard Java statements (rather than SQL statements) to change, add, or delete rows in the result cache. You can propagate changes to the cache in the underlying relational table.

Querying parameter support

The base SQL query is defined as a Java String, with parameters replacing some of the actual values. When the query runs, the data access beans provide a way to replace the parameters with values made available at run time. Default mappings for common data types are provided, but you can specify whatever your Java program and database require.

Supporting metadata

A *StatementMetaData* object contains the base SQL query. Information about the query (*metadata*) enables the object to pass parameters into the query as Java data types.

Metadata in the object maps Java data types to SQL data types (as well as the reverse). When the query runs, the Java-datatyped parameters are automatically converted to SQL data types as specified in the metadata mapping.

When results return, the metadata object automatically converts SQL data types back into the Java data types specified in the metadata mapping.

Connection management architecture

The connection management architecture for both relational and procedural access to enterprise information systems (EIS) is based on the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification. The Connection Manager (CM), which pools and manages connections within an application server, is capable of managing connections obtained through both resource adapters (RAs) defined by the JCA specification, and data sources defined by the Java Database Connectivity (JDBC) 2.0 (and later) Extensions specification.

To make data source connections manageable by the CM, the WebSphere Application Server provides a resource adapter (the WebSphere Relational Resource Adapter) that enables JDBC data sources to be managed by the same CM that manages JCA connections. From the CM point of view, JDBC data sources and JCA connection factories look the same. Users of data sources do not experience any programmatic or behavioral differences in their applications because of the underlying JCA architecture. JDBC users still configure and use data sources according to the JDBC programming model.

Applications migrating from previous versions of WebSphere Application Server might experience some behavioral differences because of the specification changes from various Java EE requirements levels. These differences are not related to the adoption of the JCA architecture.

If you have Java 2 Platform, Enterprise Edition (J2EE) 1.2 applications using the JDBC API that you wish to run in WebSphere Application Server 6.0 and later, the JDBC CM from Application Server version 4.0 is still provided as a configuration option. Using this configuration option enables J2EE 1.2 applications to run unaltered. If you migrate a Version 4.0 application to Version 6.0 or later, using the latest migration tools, the application automatically uses the Version 4.0 connection manager after migration. However, EJB 2.x modules in J2EE 1.3, J2EE 1.4 and Java Platform, Enterprise Edition (Java EE) applications cannot use the JDBC CM from WebSphere Application Server Version 4.0.

Connection pooling

Using connection pools helps to both alleviate connection management overhead and decrease development tasks for data access.

Each time an application attempts to access a backend store (such as a database), it requires resources to create, maintain, and release a connection to that datastore. To mitigate the strain this process can place on overall application resources, the Application Server enables administrators to establish a pool of backend connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

The application server supports JDBC 4.0 APIs for connection pooling and connection reuse. The connection pool is used to direct JDBC calls within the application, as well as for enterprise beans using the database.

Benefits of connection pooling

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a data source. Because users connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total datastore overhead quickly becomes high for Web-based applications, and performance deteriorates. When connection pooling capabilities are used, however, Web applications can realize performance improvements of up to 20 times the normal results.

With connection pooling, most user requests do not incur the overhead of creating a new connection because the data source can locate and use an existing connection from the pool of connections. When the request is satisfied and the response is returned to the user, the resource returns the connection to the connection pool for reuse. The overhead of a disconnection is avoided. Each user request incurs a fraction of the cost for connecting or disconnecting. After the initial resources are used to produce the connections in the pool, additional overhead is insignificant because the existing connections are reused.

When to use connection pooling

Use connection pooling in an application that meets any of the following criteria:
- It cannot tolerate the overhead of obtaining and releasing connections whenever a connection is used.
- It requires Java Transaction API (JTA) transactions within the Application Server.
- It needs to share connections among multiple users within the same transaction.
- It needs to take advantage of product features for managing local transactions within the application server.
- It does not manage the pooling of its own connections.
- It does not manage the specifics of creating a connection, such as the database name, user name, or password

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

How connections are pooled together

When you configure a unique data source or connection factory, you must give it a unique Java Naming and Directory Interface (JNDI) name. This JNDI name, along with its configuration information, is used to create the connection pool. A separate connection pool exists for each configured data source or connection factory.

Furthermore, the application server creates a separate instance of the connection pool in each application server that uses the data source or connection factory. For example:
- If you run a three server cluster in which all of the servers use *myDataSource*, and *myDataSource* has a Maximum Connections setting of 10, then you can generate up to 30 connections (three servers times 10 connections).

Consider how this behavior potentially impacts the number of connections that your backend resource can support. See “Connection pool settings” on page 1400 for more information.

Other considerations for determining the maximum connections setting:

- Each entity bean transaction requires an additional database connection, dedicated to handling the transaction.
- If clones are used, one data pool exists for each clone.

It is also important to note that when using connection sharing, it is only possible to share connections obtained from the same connection pool.

Connection and connection pool statistics:

WebSphere Application Server supports use of PMI APIs to monitor the performance of data access applications.

Performance Monitoring Infrastructure (PMI) method calls that are supported in the two existing Connection Managers (JDBC and J2C) are supported in this version of WebSphere Application Server. The calls include:

- ManagedConnectionsCreated
- ManagedConnectionsAllocated
- ManagedConnectionFreed
- ManagedConnectionDestroyed
- BeginWaitForConnection
- EndWaitForConnection
- ConnectionFaults
- Average number of ManagedConnections in the pool
- Percentage of the time that the connection pool is using the maximum number of ManagedConnections
- Average number of threads waiting for a ManagedConnection
- Average percent of the pool that is in use
- Average time spent waiting on a request
- Number of ManagedConnections that are in use
- Number of Connection Handles
- FreePoolSize
- UseTime

Java Specification Request (JSR) 77 requires statistical data to be accessed through managed beans (Mbeans) to facilitate this. The Connection Manager passes the ObjectNames of the Mbeans created for this pool. In the case of Java Message Service (JMS) *null* is passed in. The interface used is:

```
PmiFactory.createJ2CPerf(
    String pmiName, // a unique Identifier for JCA /JDBC.  This is the
                    // ConnectionFactory name.

    ObjectName providerName,// the ObjectName of the J2CResourceAdapter
                             // or JDBCProvider Mbean

    ObjectName factoryName // the ObjectName of the J2CConnectionFactory
                           // or DataSourceMbean.
)
```

The following Unified Modeling Language (UML) diagram shows how JSR 77 requires statistics to be reported:

JCAConnectionPoolStats and JDBCConnectionPoolStats objects do not have a direct implementing Mbean; the statistics are gathered through a call to PMI. A J2C resource adapter, and JDBC provider each contain a list of ConnectionFactory or DataSource ObjectNames, respectively. The ObjectNames are used by PMI to find the appropriate connection pool in the list of PMI modules.

The JCA 1.5 Specification allows an exception from the matchManagedConnection() method that indicates that the resource adapter requests that the connection not be pooled. In that case, statistics for that connection are provided separately from the statistics for the connection pool.

Connection life cycle

A ManagedConnection object is always in one of three states: *DoesNotExist*, *InFreePool*, or *InUse*.

Before a connection is created, it must be in the DoesNotExist state. After a connection is created, it can be in either the InUse or the InFreePool state, depending on whether it is allocated to an application.

Between these three states are *transitions*. These transitions are controlled by *guarding conditions*. A guarding condition is one in which *true* indicates when you can take the transition into another legal state. For example, you can make the transition from the InFreePool state to InUse state only if:

- the application has called the data source or connection factory getConnection() method (the *getConnection* condition)
- a free connection is available in the pool with matching properties (the *freeConnectionAvailable* condition)
- and one of the two following conditions are true:
  - the getConnection() request is on behalf of a resource reference that is marked unsharable
  - the getConnection() request is on behalf of a resource reference that is marked shareable but no shareable connection in use has the same properties.

This transition description follows:

InFreePool > InUse:
getConnection AND
freeConnectionAvailable AND
NOT(shareableConnectionAvailable)

Here is a list of guarding conditions and descriptions.

Condition	Description
ageTimeoutExpired	Connection is older then its ageTimeout value.
close	Application calls close method on the Connection object.
fatalErrorNotification	A connection has just experienced a fatal error.
freeConnectionAvailable	A connection with matching properties is available in the free pool.
getConnection	Application calls getConnection method on a data source or connection factory object.
markedStale	Connection is marked as stale, typically in response to a fatal error notification.
noOtherReferences	There is only one connection handle to the managed connection, and the Transaction Service is not holding a reference to the managed connection.
noTx	No transaction is in force.
poolSizeGTMin	Connection pool size is greater than the minimum pool size (minimum number of connections)
poolSizeLTMax	Pool size is less than the maximum pool size (maximum number of connections)
shareableConnectionAvailable	The getConnection() request is for a shareable connection, and one with matching properties is in use and available to share.
TxEnds	The transaction has ended.
unshareableConnectionRequest	The getConnection() request is for an unshareable connection.
unusedTimeoutExpired	Connection is in the free pool and not in use past its unused timeout value.

Getting connections

The first set of transitions covered are those in which the application requests a connection from either a data source or a connection factory. In some of these scenarios, a new connection to the database results. In others, the connection might be retrieved from the connection pool or shared with another request for a connection.

DoesNotExist

Every connection begins its life cycle in the DoesNotExist state. When an application server starts, the connection pool does not exist. Therefore, there are no connections. The first connection is not created until an application requests its first connection. Additional connections are created as needed, according to the guarding condition.

```
getConnection AND
NOT(freeConnectionAvailable) AND
poolSizeLTMax AND
(NOT(shareableConnectionAvailable) OR
unshareableConnectionRequest)
```

This transition specifies that a connection object is not created unless the following conditions occur:
- The application calls the getConnection() method on the data source or connection factory
- No connections are available in the free pool (NOT(freeConnectionAvailable))
- The pool size is less than the maximum pool size (poolSizeLTMax)
- If the request is for a sharable connection and there is no sharable connection already in use with the same sharing properties (NOT(shareableConnectionAvailable)) OR the request is for an unsharable connection (unshareableConnectionRequest)

All connections begin in the DoesNotExist state and are only created when the application requests a connection. The pool grows from 0 to the maximum number of connections as applications request new connections. The pool is **not** created with the minimum number of connections when the server starts.

If the request is for a sharable connection and a connection with the same sharing properties is already in use by the application, the connection is shared by two or more requests for a connection. In this case, a new connection is not created. For users of the JDBC API these sharing properties are most often *userid/password* and *transaction context*; for users of the Resource Adapter Common Client Interface (CCI) they are typically *ConnectionSpec*, *Subject*, and *transaction context*.

InFreePool

The transition from the InFreePool state to the InUse state is the most common transition when the application requests a connection from the pool.

```
InFreePool>InUse:
getConnection AND
freeConnectionAvailable AND
(unshareableConnectionRequest OR
NOT(shareableConnectionAvailable))
```

This transition states that a connection is placed in use from the free pool if:
- the application has issued a getConnection() call
- a connection is available for use in the connection pool (freeConnectionAvailable),
- and one of the following is true:
  - the request is for an unsharable connection (unsharableConnectionRequest)
  - no connection with the same sharing properties is already in use in the transaction. (NOT(sharableConnectionAvailable)).

Any connection request that a connection from the free pool can fulfill does not result in a new connection to the database. Therefore, if there is never more than one connection used at a time from the pool by any number of applications, the pool never grows beyond a size of one. This number can be less than the minimum number of connections specified for the pool. One way that a pool grows to the minimum number of connections is if the application has multiple concurrent requests for connections that must result in a newly created connection.

InUse

The idea of connection sharing is seen in the transition on the InUse state.

```
InUse>InUse:
getConnection AND
ShareableConnectionAvailable
```

This transition indicates that if an application requests a shareable connection (getConnection) with the **same** sharing properties as a connection that is already in use (ShareableConnectionAvailable), the existing connection is shared.

The same user (*user name* and *password*, or *subject*, depending on authentication choice) can share connections but only within the same transaction and only when all of the sharing properties match. For JDBC connections, these properties include the *isolation level*, which is configurable on the

resource-reference (IBM WebSphere extension) to data source default. For a resource adapter factory connection, these properties include those specified on the ConnectionSpec object. Because a transaction is normally associated with a single thread, you should **never** share connections across threads.

Note: It is possible to see the same connection on multiple threads at the same time, but this situation is an error state usually caused by an application programming error.

Returning connections

All of the transitions discussed previously involve getting a connection for application use. With that goal, the transitions result in a connection closing, and either returning to the free pool or being destroyed. Applications should explicitly close connections (note: the connection that the user gets back is really a connection handle) by calling close() on the connection object. In most cases, this action results in the following transition:

```
InUse>InFreePool:
(close AND
noOtherReferences AND
NoTx AND
UnshareableConnection)
OR
(ShareableConnection AND
TxEnds)
```

Conditions that cause the transition from the InUse state are:

- If the application or the container calls close() (producing the close condition) and there are no references (the noOtherReferences condition) either by the application (in the application sharing condition) or by the transaction manager (in the NoTx condition, meaning that the transaction manager holds a reference when the connection is enlisted in a transaction), the connection object returns to the free pool.
- If the connection was enlisted in a transaction but the transaction manager ends the transaction (the txEnds condition), and the connection was a shareable connection (the ShareableConnection condition), the connection closes and returns to the pool.

When the application calls close() on a connection, it is returning the connection to the pool of free connections; it is **not** closing the connection to the data store. When the application calls close() on a currently shared connection, the connection is *not returned* to the free pool. Only after the application drops the last reference to the connection, and the transaction is over, is the connection returned to the pool. Applications using unsharable connections must take care to close connections in a timely manner. Failure to do so can starve out the connection pool, making it impossible for any application running on the server to get a connection.

When the application calls close() on a connection enlisted in a transaction, the connection is not returned to the free pool. Because the transaction manager must also hold a reference to the connection object, the connection cannot return to the free pool until the transaction ends. Once a connection is enlisted in a transaction, you cannot use it in any other transaction by any other application until after the transaction is complete.

There is a case where an application calling close() can result in the connection to the data store closing and bypassing the connection return to the pool. This situation happens if one of the connections in the pool is considered stale. A connection is considered stale if you can no longer use it to contact the data store. For example, a connection is marked stale if the data store server is shut down. When a connection is marked as stale, the entire pool is cleaned out by default because it is very likely that all of the connections are stale for the same reason (or you can set your configuration to clean just the failing connection). This cleansing includes marking all of the currently InUse connections as stale so they are destroyed upon closing. The following transition states the behavior on a call to close() when the connection is marked as stale:

```
InUse>DoesNotExist:
close AND
markedStale AND
NoTx AND
noOtherReferences
```

This transition states that if the application calls close() on the connection and the connection is marked as stale during the pool cleansing step (markedStale), the connection object closes to the data store and is not returned to the pool.

Finally, you can close connections to the data store and remove them from the pool.

This transition states that there are three cases in which a connection is removed from the free pool and destroyed.

1. If a fatal error notification is received from the resource adapter (or data source). A fatal error notification (FatalErrorNotification) is received from the resource adaptor when something happens to the connection to make it unusable. All connections currently in the free pool are destroyed.
2. If the connection is in the free pool for longer than the unused timeout period (UnusedTimeoutExpired) and the pool size is greater than the minimum number of connections (poolSizeGTMin), the connection is removed from the free pool and destroyed. This mechanism enables the pool to shrink back to its minimum size when the demand for connections decreases.
3. If an age timeout is configured and a given connection is older than the timeout. This mechanism provides a way to recycle connections based on age.

Unshareable and shareable connections

The application server supports both unshareable and shareable connections. An unshareable connection is not shared with other components in the application. The component using this connection has full control of this connection.

Access to a resource marked as unshareable means that there is a one-to-one relationship between the connection handle a component is using and the physical connection with which the handle is associated. This access implies that every call to the getConnection method returns a connection handle solely for the requesting user. Typically, you must choose unshareable if you might do things to the connection that could result in unexpected behavior occurring in another application that is sharing the connection (for example, unexpectedly changing the isolation level).

Marking a resource as shareable allows for greater scalability. Instead of creating new physical connections on every getConnection() invocation, the physical connection (that is, managed connection) is shared through multiple connection handles, as long as each getConnection request has the same connection properties. However, sharing a connection means that each user must not do anything to the connection that could change its behavior and disrupt a sharing partner (for example, changing the isolation level). The user also cannot code an application that assumes sharing to take place because it is up to the run time to decide whether or not to share a particular connection.

Connection property requirements

To permit sharing of connections used within the same transaction, the following data source properties must be the same:

- Java Naming and Directory Interface (JNDI) name. While not actually a connection property, this requirement simply means that you can only share connections from the same data source in the same server.
- Resource authentication
- In relational databases:
  - Isolation level (corresponds to access intent policies applied to CMP beans)
  - Readonly
  - Catalog
  - TypeMap

For more information on sharing a connection with a CMP bean, see “Sharing a connection with a CMP bean.”

To permit sharing of connections within the same transaction, the following properties must be the same for the connection factories:
- JNDI name. While not actually a connection property, this requirement simply means that you can only share connections from the same connection factory in the same server.
- Resource authentication

In addition, the ConnectionSpec object that is used to get the connection must also be the same.

Note: Java Message Service (JMS) connections cannot be shared with non-JMS connections.

Sharing a connection with a CMP bean

The application server allows you to share a physical connection among a CMP bean, a BMP bean, and a JDBC application to reduce the resource allocation or deadlock scenarios. There are several ways to ensure that all of these entity beans and the JDBC applications are sharing the same physical connection.

- **Sharing a connection between CMP beans or methods**

  When all CMP bean methods use the same access intent, they all share the same physical connection. A different access intent policy triggers the allocation of a different physical connection. For example, a CMP bean has two methods; method 1 is associated with `wsPessimisticUpdate` intent, whereas method 2 has `wsOptimisticUpdate` access intent. Method 1 and method 2 cannot share the same physical connection within a transaction. In other words, an XA data source is required to run in a global transaction.

  You can experience some deadlocks from a database if both methods try to access the same table. Therefore, sharing a connection is determined by the access intents that are defined in the CMP methods.

- **Sharing a connection between CMP and BMP beans**

  Remember to first verify that the **getConnection** methods of both the BMP bean and the CMP bean set the same connection properties. To match the authentication type of the CMP bean resource, set the authentication type of the BMP bean resource to container-managed, which is designated in the deployment descriptor as `res-auth = Container`.

  Additionally, use one of the following options to ensure connection-sharing between the bean types:

  – Define the same access intent on both CMP and BMP bean methods. Because both use the same access intent, they share the same physical connection. The advantage to using this option is that the backend is transparent to a BMP bean. However, this option also makes the BMP non-portable because it uses the WebSphere extended API to handle the isolation level. For more information, refer to the code example in Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans.

  – Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level that is specified on the resource reference to look up a data source and a connection. This option is more of a manual process, and the isolation level might be different from database to database. For more information refer to the isolation level and access intent mapping table: Access intent isolation levels and update locks and the Isolation level and resource reference section.

- **Sharing a connection between CMP and a JDBC application that is used by a servlet or a session bean**

  Determine the isolation level that the access intent uses on a CMP bean method, then use the corresponding isolation level specified on the resource reference to look up a data source and a connection. For more information refer to Access intent isolation levels and update locks and Isolation level and resource reference.

Factors that determine sharing

The listing here is not an exhaustive one. The product might or might not share connections under different circumstances.

- Only connections acquired with the same resource reference (resource-ref) that specifies the res-sharing-scope as shareable are candidates for sharing. The resource reference properties of res-sharing-scope and res-auth and the IBM extension isolationLevel help determine if it is possible to share a connection. IBM extension isolationLevel is stored in IBM deployment descriptor extension file; for example: ibm-ejb-jar-ext.xmi.
- You can only share connections that are requested with the same properties.
- Connection sharing only occurs between different component instances if they are within a transaction (container- or user-initiated transaction).
- Connection sharing only occurs within a sharing boundary. Current® sharing boundaries include *Transactions* and *LocalTransactionContainment* (LTC) boundaries.
- Connection sharing rules within an LTC Scope:
  - For shareable connections, only *Connection Reuse* is allowed within a single component instance. Connection reuse occurs when the following actions are taken with a connection: get, use, commit/rollback, close; get, use, commit/rollback, close. Note that if you use the LTC resolution-control of *ContainerAtBoundary* then no start/commit is needed because that action is handled by the container.

    The connection returned on the second *get* is the same connection as that returned on the first *get* (if the same properties are used). Because the connection use is serial, only one connection handle to the underlying physical connection is used at a time, so true connection sharing does not take place. The term *"reuse"* is more accurate.

    More importantly, the *LocalTransactionContainment* boundary enclosing both *get* actions is not complete; no cleanUp() method is invoked on the ManagedConnection object. Therefore the second *get* action inherits all of the connection properties set during the first getConnection() call.
- Shareable connections between transactions (either container-managed transactions (CMT), bean-managed transactions (BMT), or LTC transactions) follow these caching rules:
  - In general, setting properties on shareable connections is not allowed because a user of one connection handle might not anticipate a change made by another connection handle. This limitation is part of the Java Platform, Enterprise Edition (Java EE) standard.
  - General users of resource adapters can set the connection properties on the connection factory getConnection() call by passing them in a ConnectionSpec.

    However, the properties set on the connection during one transaction are not guaranteed to be the same when used in the next transaction. Because it is not valid to share connections outside of a sharing scope, connection handles are moved off of the physical connection with which they are currently associated when a transaction ends. That physical connection is returned to the free connection pool. Connections are cleaned before going in the free pool. The next time the handle is used, it is automatically associated with an appropriate connection. The appropriateness is based on the security login information, connection properties, and (for the JDBC API) the isolation level specified in the extended resource reference, passed in on the original request that returned the current handle. Any properties set on the connection after it was retrieved are lost.
  - For JDBC users, the application server provides an extension to enable passing the connection properties through the ConnectionSpec.

    Use caution when setting properties and sharing connections in a local transaction scope. Ensure that other components with which the connection is shared are expecting the behavior resulting from your settings.
- You cannot set the isolation level on a shareable connection for the JDBC API using a relational resource adapter in a global transaction. The product provides an extension to the resource reference to enable you to specify the isolation level. If your application requires the use of multiple isolation levels, create multiple resource references and map them to the same data source or connection factory.

Maximal connection sharing

To maximize connection sharing opportunities for an application, ensure that each component has the local transaction containment (LTC) Resolver attribute set to **ContainerAtBoundary**. This setting specifies that the component container, rather than the application code, resolves all resource manager local transactions (RMLTs) within the LTC scope. The container begins an RMLT when a connection is first used within the LTC scope, and completes it automatically at the end of the LTC scope.

Consult the topic Configuring transactional deployment attributes for instructions on setting the transaction resolution control and other attributes.

Connection sharing violations

There is a new exception, the sharing violation exception, that the resource adapter can issue whenever an operation violates sharing requirements. Possible violations include changing connection attributes, security settings, or isolation levels, among others. When such a mutable operation is performed against a managed connection, the sharing violation exception can occur when both of the following conditions are true:

- The number of connection handles associated with the managed connection is more than one.
- The managed connection is associated with a transaction, either local or XA.

Both the component and the J2C run time might need to detect this sharing violation exception, depending on when and how the managed connection becomes unshareable. If the managed connection becomes unshareable because of an operation through the connection handle (for example, you change the isolation level), then the component needs to process the exception. If the managed connection becomes unshareable without being recognized by the application server (due to some component interaction with the connection handle), then the resource adapter can reject the creation of a connection handle by issuing the sharing violation exception.

Connection handles

A connection handle is a representation of a physical connection. To use a backend resource (such as a relational database) in WebSphere Application Server you must get a connection to that resource. When you call the *getConnection()* method, you get a *connection handle* returned. The handle is not the physical connection. The physical connection is managed by the connection manager.

There are two significant configurations that affect how connection handles are used and how they behave. The first is the *res-sharing-scope*, which is defined by the resource-reference used to look up the DataSource or Connection Factory. This property tells the connection manager whether or not you can share this connection.

The second factor that affects connection handle behavior is the *usage pattern*. There are essentially two usage patterns. The first is called the *get/use/close* pattern. It is used within a single method and without calling another method that might get a connection from the same data source or connection factory. An application using this pattern does the following:

1. gets a connection
2. does its work
3. commits (if appropriate)
4. closes the connection.

The second usage pattern is called the *cached handle* pattern. This is where an application:

1. gets a connection
2. begins a global transaction
3. does work on the connection
4. commits a global transaction
5. does work on the connection again

A cached handle is a connection handle that is held across transaction and method boundaries by an application. Keep in mind the following considerations for using cached handles:

- Cached handle support requires some additional connection handle management across these boundaries, which can impact performance. For example, in a JDBC application, *Statements*, *PreparedStatements*, and *ResultSets* are closed implicitly after a transaction ends, but the connection remains valid.
- You are encouraged **not** to cache the connection across the transaction boundary for shareable connections; the get/use/close pattern is preferred.
- Caching of connection handles across servlet methods is limited to JDBC and Java Message Service (JMS) resources. Other non-relational resources, such as Customer Information Control System (CICS) or IMS objects, currently cannot have their connection handles cached in a servlet; you need to get, use, and close the connection handle within each method invocation. (This limitation only applies to single-threaded servlets because multithreaded servlets do not allow caching of connection handles.)
- You **cannot** pass a cached connection handle from one instance of a data access client to another client instance. Transferring between client instances creates the problematic contingency of one instance using a connection handle that is referenced by another. This relationship can only cause problems because connection handle management code processes tasks for each client instance *separately*. Hence, connection handle transfers result in run-time scenarios that trigger exceptions. For example:
  1. The application code of a client instance that receives a transferred handle closes the handle.
  2. If the client instance that retains the original reference to the handle tries to reclaim it, the application server issues an exception.

The following code segment shows the cached connection pattern.

```
Connection conn = ds.getConnection();
ut.begin();
conn.prepareStatement("....."); --> Connection runs in global transaction mode
...
ut.commit();
conn.prepareStatement(".....");   ---> Connection still valid but runs in autoCommit(True);
...
```

Unshareable connections

Some characteristics of connection handles retrieved with a *res-sharing-scope* of **unshareable** are described in the following sections.

- **The possible benefits of unshared connections**
  - Your application always maintains a direct link with a physical connection (managed connection).
  - The connection always has a one-to-one relationship between the connection handle and the managed connection.
  - In most cases, the connection does not close until the application closes it.
  - You can use a cached unshared connection handle across multiple transactions.
  - The connection can have a performance advantage in some cached handle situations. Because unshared connections do not have the overhead of moving connection handles off managed connections at the end of the transaction, there is less overhead in using a cached unshared connection.
- **The possible drawbacks of unshared connections**
  - Inefficient use of your connection resources. For example, if within a single transaction you get more than one connection (with the same properties) using the same data source or connection factory (same resource-ref) then you use multiple physical connections when you use unshareable connections.
  - Wasted connections. It is important not to keep the connection handle open (that is, your application does not call the *close()* method) any longer then it is needed. As long as an unshareable connection

is open, the physical connection is unavailable to any other component, even if your application is not currently using that connection. Unlike a shareable connection, an ushareable connection is not closed at the end of a transaction or servlet call.

– Deadlock considerations. Depending on how your components interact with the database within a transaction, using unshared connections can lead to deadlock in the database. For example, within a transaction, component A gets a connection to data source X and updates table 1, and then calls component B. Component B gets another connection to data source X, and updates/reads table 1 (or even worse the same row as component A). In some circumstances, depending on the particular database, its locking scheme, and the transaction isolation level, a deadlock can occur.

  In the same scenario, but with a *shared* connection, deadlock does not occur because all the work is done on the same connection. It is worth noting that when writing code that uses shared connections, you use a strategy that calls for multiple work items to be performed on the same connection, possibly within the same transaction. If you decide to use an unshareable connection, you must set the *maximum connections* property on the connection factory or data source correctly. An exception might occur for waiting connection requests if you exceed the maximum connections value, and unshareable connections are not being closed before the connection wait time-out is exceeded.

Shareable connections

Some characteristics of connection handles that are retrieved with a *res-sharing-scope* of **shareable** are described in the following sections.

- **The possible benefits of shared connections**
  – Within an instance of connection sharing, application components can share a managed connection with one or more connection handles, depending on how the handle is retrieved and which connection properties are used.
  – They can more efficiently use resources. Shareable connections are not valid outside of their sharing boundary. For this reason, at the end of a sharing boundary (such as a transaction) the connection handle is no longer associated with the managed connection it was using within the sharing boundary (this applies only when using the cached handle pattern). The managed connection is returned to the free connection pool for reuse. Connection resources are not held longer than the end of the current sharing scope.

    If the cached handle pattern is used, then the next time the handle is used within a new sharing scope, the application server run time ensures that the handle is reassociated with a managed connection that is appropriate for the current sharing scope, and has the same properties with which the handle was originally retrieved. Remember that it is not appropriate to change properties on a shareable connection. If properties are changed, other components that share the same connection might experience unexpected behavior. Futhermore, when using cached handles, the value of the changed property might not be remembered across sharing scopes.
- **The possible drawbacks of shared connections**
  – Sharing within a single component (such as an enterprise bean and its related Java objects) is not always supported. The current specification allows resource adapters the choice of only allowing one active connection handle at a time.

    If a resource adapter chooses to implement this option then the following scenario results in an *invalid handle exception*: A component using shareable connections gets a connection and uses it. Without closing the connection, the component calls a utility class (Java object) that gets a connection handle to the same managed connection and uses it. Because the resource adapter only supports one active handle, the first connection handle is no longer valid. If the utility object returns without closing its handle, the first handle is not valid and triggers an exception at any attempt to use it.

    Note: This exception occurs only when calling a utility object (a Java object).

    Not all resource adapters have this limitation; it occurs only in certain implementations. The WebSphere Relational Resource Adapter (RRA) does not have this limitation. Any data source used

through the RRA does not have this limitation. If you encounter a resource adapter with this limitation you can work around it by serializing your access to the managed connection. If you always close your connection handle before getting another (or close your handle before calling code that gets another handle), and before returning from a method, you can allow two pieces of code to share the same managed connection. You simply cannot use the connection for both events at the same time.
- Trying to change the *isolation level* on a shareable JDBC-based connection in a global transaction (that is supported by the RRA) causes an exception. The correct way to get connections with different transaction isolation levels is by configuring the IBM extended resource-reference.
- Closing connection handles for shareable connections by an application is NOT supported and causes errors. However, you can avoid this limitation by using the Relational Resource Adapter.

Lazy connection association optimization

The Java Platform, Enterprise Edition (Java EE) Connector (J2C) connection manager implemented *smart handle* support. This technology enables allocation of a connection handle to an application while the managed connection associated with that connection handle is used by other applications (assuming that the connection is not being used by the original application). This concept is part of the Java EE Connector Architecture (JCA) 1.5 specification. (You can find it in the JCA 1.5 specification document in the section entitled ″Lazy Connection Association Optimization.″) Smart handle support introduces use a method on the ConnectionManager object, the *LazyAssociatableConnectionManager()* method, and a new marker interface, the *DissociatableManagedConnection* class. You must configure the provider of the resource adapter to make this functionality available in your environment. (In the case of the RRA, WebSphere Application Server itself is the provider.) The following code snippet shows how to include smart handle support:

```
package javax.resource.spi;
import javax.resource.ResourceException;

interface LazyAssociatableConnectionManager { // application server
    void associateConnection(
        Object connection, ManagedConnectionFactory mcf,
        ConnectionRequestInfo info) throws ResourceException;
}

interface DissociatableManagedConnection { // resource adapter
    void dissociateConnections() throws ResourceException;
}
```

This DissociatableManagedConnection interface introduces another state to the Connection object: *inactive*. A Connection can now be active, closed, and inactive. The connection object enters the inactive state when a corresponding ManagedConnection object is cleaned up. The connection stays inactive until an application component attempts to re-use it. Then the resource adapter calls back to the connection manager to re-associate the connection with an active ManagedConnection object.

Transaction type and connection behavior

All connection usage occurs within the scope of either a global transaction or a local transaction containment (LTC) boundary. Each transaction type places different requirements on connections and impacts connection settings differently.

Connection sharing and reuse

You can share connections within a global transaction scope (assuming other sharing rules are met). You can also share connections within a shareable LTC. You can *serially reuse* connections within an LTC scope. A get/use/close connection pattern followed by another instance of get/use/close (to the same data source or connection factory) enables you to reuse the same connection. See the “Unshareable and shareable connections” on page 1328 topic for more details.

JDBC AutoCommit behavior

All JDBC connections, when first obtained through a getConnection() call, contain the setting AutoCommit = TRUE by default. However, different transaction scope and settings can result in changing, or simply overriding, the AutoCommit value.

- If you operate within an LTC and have its resolution-control set to *Application*, AutoCommit remains *TRUE* unless changed by the application.
- If you operate within an LTC and have its resolution-control set to *ContainerAtBoundary*, the application must **not** touch the AutoCommit setting. The WebSphere Application Server run time sets the AutoCommit value to *FALSE* before work begins, then commits or rolls back the work, as appropriate, at the end of the LTC scope.
- If you use a connection within a global transaction, the database ignores the AutoCommit setting so that the transaction service that controls the commit and rollback processing can manage the transaction. This action takes place upon first use of the connection to do work, regardless of the user changing the AutoCommit setting. After the transaction completes, the AutoCommit value returns to the value it had before the first use of the connection. So even if the AutoCommit value is set to *TRUE* before the connection is used in a global transaction, you need not set the value to *FALSE* because the value is ignored by the database. In this example, after the transaction completes, the AutoCommit value of the connection returns to *TRUE*.
- If you use multiple distinct connections within a global transaction, all work is guaranteed to commit or roll back together. This is not the case for a local transaction containment (LTC scope). Within an LTC, work done on one connection commits or rolls back independently from work done on any other connection within the LTC.

One-phase commit and two-phase commit connections

The type and number of resource managers, such as a database server, that must be accessed by an application often determines the application transaction requirements. Consequently each type of resource manager places different requirements on connection behavior.

- A two-phase commit resource manager can support two-phase coordination of a transaction. That support is necessary for transactions that involve other resource managers; these transactions are global transactions. See “Transaction support in WebSphere Application Server” on page 2125 for further explanation.
- A one-phase commit resource manager supports only one-phase transactions, or LTC transactions, in which that resource is the sole participating datastore. Again, see “Transaction support in WebSphere Application Server” on page 2125 for further explanation.

One-phase commit resources are such that work being done on a one phase connection cannot mix with other connections and ensure that the work done on all of the connections completes or fails atomically. The product does not allow more than one one-phase commit connection in a global transaction. Futhermore, it does not allow a one-phase commit connection in a global transaction with one or more two-phase commit connections. You can coordinate only multiple two-phase commit connections within a global transaction.

WebSphere Application Server provides *last participant support*, which enables a single one-phase commit resource to participate in a global transaction with one or more two-phase commit resources.

Note that any time that you do multiple getConnection() calls using a resource reference that specifies res-sharing-scope=Unshareable, you get multiple physical connections. This situation also occurs when res-sharing-scope=Shareable, but the sharing rules are broken. In either case, if you run in a global transaction, ensure the resources involved are enabled for two-phase commit (also sometimes referred to as *JTA Enabled*). Failure to do so results in an XA exception that logs the following message:

```
WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable
resources has occurred.
```

Application scoped resources

Use this page to view brief descriptions of the resources that are bundled with your application. You can view individual resource settings by clicking on the resource name.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Application scoped resources**.

Each table row corresponds to a resource that is bundled with your application. Click a resource name or the corresponding provider name to view an administrative console page where you can edit the object configuration settings.

Name:

Specifies the administrative name that was assigned to this resource.

Click this name to view a page where you can edit the configuration settings.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of the resource.

Data type	String

Resource type:

Specifies the type of resource, such as a data source or a J2C connection factory.

Provider:

Specifies the resource provider that supplies the class information for this resource object.

Click the provider name to view a page where you can edit the configuration settings.

Description:

Specifies a text description of the resource.

Cache instances

An application uses a cache instance to store, retrieve, and share data objects within the dynamic cache.

Each cache instance can be configured independently for Java Naming and Directory Interface (JNDI) name, cache size, priority, and disk offload. Objects that are stored in a particular cache instance are not affected by other cache instances. This means that if you store an object named **object_1** with a value of `object_data` in *cache_instance_x*, you can also store an object with the same name, but different value in *cache_instance_y*.

Objects that are stored in a particular cache instance are available to applications on other servers by accessing a cache instance of the same name. The two servers must be within the same replication domain to share data.

There are two types of cache instances, object cache instances and servlet cache instances.

An object cache instance is a location in addition to the default shared dynamic cache where Java 2 Platform, Enterprise Edition (J2EE) applications can store, distribute, and share objects. After configuring

object cache instances, you can use the DistributedMap or DistributedObjectCache interfaces in the `com.ibm.websphere.cache` package to programmatically access your cache instances.

See the Additional Application Programming Interfaces (APIs) for more information about the DistributedMap or DistributedObjectCache interfaces.

Servlet cache instances are locations in addition to the default dynamic cache where dynamic cache can store, distribute, and share the output and the side effects of an invoked servlet. By configuring a servlet cache instance, your applications have greater flexibility and better tuning of cache resources. The Java Naming and Directory Interface (JNDI) name that is specified for the cache instance in the administrative console maps to the <cache-instance> element in the `cachespec.xml` configuration file. Any <cache-entry> elements that are specified within a <cache-instance> element are created in that specific cache instance. Any <cache-entry> elements that are specified outside of a <cache-instance> element are stored in the default dynamic cache instance.

See “Using servlet cache instances” on page 2277 for more information.

Data access: Resources for learning

Use the following links to find relevant supplemental information about data access. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product, but it can be useful for understanding concepts or functions used by the application server. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information:

Technologies for data access

- JDBC 3.0 API Documentation
- **Java Persistence API:**
  - Java Persistence API FAQ
  - Introduction to Spring 2 and JPA
- J2EE Connector Architecture Version 1.5 specification
- Enterprise JavaBeans Technology (Source for download of the Enterprise Javabeans 3.0 specification)
- Java 2 Platform, Enterprise Edition (J2EE)
- **Container-managed relationships:** Enterprise JavaBeans 2.0 Container-Managed Persistence Example. Although this article addresses the EJB 2.0 specification, you might find parts of it pertinent to your environment.
- **Resource adapters:** The J2EE Connector Architecture Resource AdapterDeveloper Technical Articles & Tips -- Articles: Database Access (Sun Developer Network)
- Java Management Extensions (JMX)
- **Miscellaneous articles from the Sun Developer Network and IBM developerWorks Web sites:**
  - Sharing connections in WebSphere Application Server V5 This article is still pertinent to WebSphere Application Server Version 6.0 and later. However, be aware that the container-managed authentication type is deprecated.

  - Database authentication in WebSphere Application Server V5 This article is still pertinent to WebSphere Application Server Version 6.0 and later. However, be aware that the container-managed authentication type is now deprecated.
  - Understanding WebSphere Application Server EJB access intents
- Supported hardware, software, and APIs

Databases

- **Cloudscape:**
  - IBM Cloudscape product information
  - IBM Cloudscape information center
- DB2 database software
- Informix

Tools

- Rational Application Developer for WebSphere Software

Configuring data access with scripting

Use these topics to learn about using scripting to configure data access.

About this task

This topic contains the following tasks:

- "Configuring a JDBC provider using scripting"
- "Configuring new data sources using scripting" on page 1340
- "Configuring new connection pools using scripting" on page 1341
- "Changing connection pool settings with the wsadmin tool" on page 1342
- "Configuring new data source custom properties using scripting" on page 1348
- "Configuring new Java 2 Connector authentication data entries using scripting" on page 1349
- "Configuring new WAS40 data sources using scripting" on page 1350
- "Configuring new WAS40 connection pools using scripting" on page 1351
- "Configuring new WAS40 custom properties using scripting" on page 1353
- "Configuring new J2C resource adapters using scripting" on page 1354
- "Configuring custom properties for J2C resource adapters using scripting" on page 1355
- "Configuring new J2C connection factories using scripting" on page 1356
- "Configuring new J2C administrative objects using scripting" on page 1360
- "Configuring new J2C activation specifications using scripting" on page 1358
- "Managing the message endpoint lifecycle using scripting" on page 1361
- "Testing data source connections using scripting" on page 1362

Configuring a JDBC provider using scripting

You can configure a JDBC provider using the wsadmin tool and scripting.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

1. There are two ways to perform this task. Perform one of the following:
   - Using the AdminTask object:

– Using Jacl:

```
$AdminTask createJDBCProvider {-interactive}
```

– Using Jython:

```
AdminTask.createJDBCProvider (['-interactive'])
```

- Using the AdminConfig object:

a. Identify the parent ID and assign it to the node variable. The following example uses the node configuration object as the parent. You can modify this example to use the cell, cluster, server, or application configuration object as the parent.

– Using Jacl:

```
set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
```

– Using Jython:

```
node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
print node
```

Example output:

```
mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
```

b. Identify the required attributes:

Note: For supported JDBC drivers, you can also script JDBC providers according to the same pre-configured templates that are used by the administrative console logic. Consult the article Creating configuration objects using the wsadmin tool for details.

– Using Jacl:

```
$AdminConfig required JDBCProvider
```

– Using Jython:

```
print AdminConfig.required('JDBCProvider')
```

Example output:

```
Attribute     Type
name     String
implementationClassName   String
```

c. Set up the required attributes and assign it to the jdbcAttrs variable. You can modify the following example to setup non-required attributes for JDBC provider.

– Using Jacl:

```
set n1 [list name JDBC1]
set implCN [list implementationClassName myclass]
set  jdbcAttrs [list  $n1  $implCN]
```

Example output:

```
{name {JDBC1}} {implementationClassName {myclass}}
```

– Using Jython:

```
n1 = ['name', 'JDBC1']
implCN = ['implementationClassName', 'myclass']
jdbcAttrs = [n1,  implCN]
print jdbcAttrs
```

Example output:

```
[['name', 'JDBC1'], ['implementationClassName', 'myclass']]
```

d. Create a new JDBC provider using node as the parent:

– Using Jacl:

```
$AdminConfig create JDBCProvider $node $jdbcAttrs
```

– Using Jython:

```
AdminConfig.create('JDBCProvider', node, jdbcAttrs)
```

Example output:

```
JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
```

2. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

What to do next

If you modify the class path or native library path of a JDBC provider: After saving your changes (and synchronizing the node in a network deployment environment), you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Configuring new data sources using scripting

You can configure new data sources using scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

In WebSphere Application Server, any JDBC driver properties that are required by your database vendor must be set as data source properties. Consult the article Vendor-specific data sources minimum required settings in the *Troubleshooting and support* PDF to see a list of these properties and setting options, ordered by JDBC provider type. Consult your database vendor documentation to learn about available optional data source properties. Script them as *custom properties* after you create the data source. In the Related links section of this article, click the ″Configuring new data source custom properties using scripting″ link for more information.

About this task

There are two ways to perform this task; use either of the following wsadmin scripting objects:

- AdminTask object
- AdminConfig object

AdminConfig gives you more configuration control than the AdminTask object. When you create a data source using AdminTask, you supply universally required properties only, such as a JNDI name for the data source. (Consult the article “JDBCProviderManagement command group for the AdminTask object” on page 1364 for more information.) Other properties that are required by your JDBC driver are assigned default values by Application Server. You cannot use AdminTask commands to set or edit these properties; you must use AdminConfig commands.

- Using the AdminConfig object to configure a new data source:
  1. Identify the parent ID, which is the name and location of the JDBC provider that supports your data source.
     - Using Jacl:

       ```
       set newjdbc [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/]
       ```

     - Using Jython:

       ```
       newjdbc = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/')
       print newjdbc
       ```

       Example output:

       ```
       JDBC1(cells/mycell/nodes/mynode|resources.xml#JDBCProvider_1)
       ```

  2. Obtain the required attributes.

     Note: For supported JDBC drivers, you can also script data sources according to the same pre-configured templates that are used by the administrative console logic. Consult the article Creating configuration objects using the wsadmin tool for details.

– Using Jacl:

```
$AdminConfig required DataSource
```

– Using Jython:

```
print AdminConfig.required('DataSource')
```

Example output:

```
Attribute   Type
name    String
```

Note: If the database vendor-required properties (which are referenced in the article "Data source minimum required settings, by vendor" on page 1433) are not displayed in the resulting list of required attributes, script these properties as data source custom properties after you create the data source.

3. Set up the required attributes.
   – Using Jacl:

```
set name [list name DS1]
set dsAttrs [list $name]
```

   – Using Jython:

```
name = ['name', 'DS1']
dsAttrs = [name]
```

4. Create the data source.
   – Using Jacl:

```
set newds [$AdminConfig create DataSource $newjdbc $dsAttrs]
```

   – Using Jython:

```
newds = AdminConfig.create('DataSource', newjdbc, dsAttrs)
print newds
```

   Example output:

```
DS1(cells/mycell/nodes/mynode|resources.xml#DataSource_1)
```

- Using the AdminTask object to configure a new data source:
  – Using Jacl:

```
$AdminTask createDatasource {-interactive}
```

  – Using Jython:

```
AdminTask.createDatasource (['-interactive'])
```

- Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

What to do next

To set additional properties that are supported by your JDBC driver, script them as data source custom properties.

Configuring new connection pools using scripting

You can use scripting and the wsadmin tool to configure new connection pools.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```
   - Using Jython:
     ```
     newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
     ```

   Example output:
   ```
   DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
   ```
2. Creating connection pool:
   - Using Jacl:
     ```
     $AdminConfig create ConnectionPool $newds {}
     ```
   - Using Jython:
     ```
     print AdminConfig.create('ConnectionPool', newds, [])
     ```

   Example output:
   ```
   (cells/mycell/nodes/mynode|resources.xml#ConnectionPool_1)
   ```
3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Changing connection pool settings with the wsadmin tool

You can use the wsadmin scripting tool to change connection pool settings.

About this task

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server run-time operations. The WebSphere Application Server only supports the Jacl and Jython scripting languages. To learn more about the wsadmin tool see Starting the wsadmin scripting client.

To use the wsadmin tool to change connection pool settings:

1. Launch a scripting command. There are several options for you to run scripting commands, ranging from running them interactively to running them in a profile.

   To change connection pool settings, you use the *getAttribute* and *setAttribute* commands, run against the various settings objects.

   For example, to change the connection timeout setting, the commands are:
   ```
   $AdminControl getAttribute $objectname connectionTimeout
   $AdminControl setAttribute $objectname connectionTimeout 200
   ```

   where:
   - $ is a Jacl operator for substituting a variable name with its value
   - getAttribute and setAttribute are the commands
   - connectionTimeout is the object whose value you are resetting
2. For Jacl code examples of each of the connection pool settings, see “Example: Changing connection pool settings with the wsadmin tool”

Example: Changing connection pool settings with the wsadmin tool

Using the wsadmin AdminControl object, you can script changes to connection pool settings.

The WebSphere Application Server wsadmin tool provides the ability to run scripts in the Jacl and Jython languages only. You must start the wsadmin scripting client to perform any scripting task. (See the article ″Starting the wsadmin scripting client.″)

For more information about the AdminControl scripting object, refer to the article ″Using the AdminControl object for scripted administration.″ See the links section at the end of this article.

Connection Timeout

The Connection timeout can be changed at any time while the pool is active. All connection requests that are waiting for a connection are changed to the new value minus the time already waited, and are returned to the wait state if there are no available connections.

For example, if the connection timeout is changed to 300 seconds and a connection request has waited 100 seconds, the connection request waits for 200 more seconds if no connection becomes available.

```
$AdminControl getAttribute $objectname connectionTimeout
$AdminControl setAttribute $objectname connectionTimeout 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Maximum Connections

The maximum connections can be changed at any time except in the case where stuck connection support is active.

If stuck connection support is active, an attempt to change the maximum connections is made. If the attempt fails, an `IllegalState` exception occurs. See the Connection pool advanced settings topic in the *Administering applications and their environment* PDF for more information.

If stuck connection support is not active, the maximum connections are changed to the new value. If the new value is greater than the current value, the number of connections increases to the new value and any requests waiting are notified. If the new value is less than the current value and Aged Timeout or Reap Time is used, the number of connections decreases to the new value depending on pool activity. If agedTimeout or Reap Time are not used, no automatic attempt will be made to reduce the total number of connections. To manually reduce the number of connections to the new maximum connections, use the Mbean function purgePoolContents.

```
$AdminControl getAttribute $objectname maxConnections
$AdminControl setAttribute $objectname maxConnections 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Minimum Connections

The minimum connections can be changed at any time

```
$AdminControl getAttribute $objectname minConnections
$AdminControl setAttribute $objectname minConnections 200
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Reap Time

The reap time can be changed at any time. The reap time interval is changed to the new value at the next interval.

```
$AdminControl getAttribute $objectname reapTime
$AdminControl setAttribute $objectname reapTime 30
```

Unused Timeout

The unused timeout can be changed at any time.

```
$AdminControl getAttribute $objectname unusedTimeout
$AdminControl setAttribute $objectname unusedTimeout 900
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Aged Timeout

The Aged Timeout can be changed at any time.

```
$AdminControl getAttribute $objectname agedTimeout
$AdminControl setAttribute $objectname agedTimeout 900
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Purge Policy

The purge policy can be changed at any time.

```
$AdminControl getAttribute $objectname purgePolicy
$AdminControl setAttribute $objectname purgePolicy "Failing Connection Only"
```

For more information about this setting, see the Connection pool settings topic in the *Administering applications and their environment* PDF.

Surge Protection Support

Surge connection support starts if surgeThreshold is > -1 and surgeCreationInterval is > 0. The surge protection properties can be changed at any time.

```
$AdminControl getAttribute $objectname surgeCreationInterval
$AdminControl setAttribute $objectname surgeCreationInterval 30
$AdminControl getAttribute $objectname surgeThreshold
$AdminControl setAttribute $objectname surgeThreshold 15
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Stuck connection Support

An attempt is made to change the stuckTime, stuckTimerTime or stuckThreshold properties. If the attempt fails, an `IllegalState` exception occurs. The pool cannot have any active requests or active connections during this request. For the stuck connection support to start, all three stuck property values must be greater than 0, and the maximum connections value must be > 0.

If the connection pool is stuck, you cannot change the stuck or the maximum connection properties. If you are stuck, there are active connections.

```
$AdminControl getAttribute $objectname stuckTime
$AdminControl setAttribute $objectname stuckTime 30
$AdminControl getAttribute $objectname stuckTimerTime
$AdminControl setAttribute $objectname stuckTimerTime 15
$AdminControl getAttribute $objectname stuckThreshold
$AdminControl setAttribute $objectname stuckThreshold 10
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Test Connection Support (This is only supported for a DataSource)

The test connection support starts when the testConnection property is set to true, and the interval is > 0. The test connection properties can be changed at any time.

```
$AdminControl getAttribute $objectname testConnection
$AdminControl setAttribute $objectname testConnection 30
$AdminControl getAttribute $objectname testConnectionInterval
$AdminControl setAttribute $objectname testConnectionInterval 15
```

For more information about this setting, see the Test connection service topic in the *Administering applications and their environment* PDF.

Connection Pool Partition Support

```
$AdminControl invoke $objectname freePoolDistributionTableSize
$AdminControl invoke $objectname numberOfFreePoolPartitions
$AdminControl invoke $objectname numberOfSharedPoolPartitions
$AdminControl invoke $objectname gatherPoolStatisticalData
$AdminControl invoke $objectname enablePoolStatisticalData
```

For more information about this setting, see the Connection pool advanced settings topic in the *Administering applications and their environment* PDF.

Show Pool Information Support

The show pool operations can be changed at any time.

```
$AdminControl invoke $objectname showAllPoolContents
$AdminControl invoke $objectname showPoolContents
$AdminControl invoke $objectname showAllocationHandleList
```

Purge Connection Support

PurgePool can be changed at any time.

```
$AdminControl invoke $objectname purgePoolContents normal
$AdminControl invoke $objectname purgePoolContents immediate
```

Pool Control Support

Pause and resume can be changed at any time.

```
$AdminControl invoke $objectname pause
$AdminControl invoke $objectname resume
```

Example: Accessing MBean connection factory and data sources using wsadmin

Wsadmin commands are used to access connection factory and data source MBeans. MBeans can retrieve or update properties for a connection factory or data source.

To get a list of J2C connection factory or data source Mbean object names, from the wsadmin command line use one of the following administrative control commands:

```
$AdminControl queryNames *:type=J2CConnectionFactory,*
$AdminControl queryNames *:type=DataSource,*
```

Example output from DataSource query:

```
wsadmin>$AdminControl queryNames *:type=DataSource,*
"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Derby JDBC Provider,j2eeType=JDBCDataSource,J2EESe
```

```
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Derby JDBC Provider,cell=system1Node01Cell"
```

By using the J2C connection factory or data source MBean object name, you can access the MBean. The name follows the `webSphere:name=` expression. In the following example, for the first set, the default data source name is set on variable name. For the second set, the object name is set on variable *objectName*.

Example using the Default Datasource

```
- wsadmin>set name [list Default Datasource]

Default Datasource


- wsadmin>set objectName [$AdminControl queryNames *:name=$name,*]


"WebSphere:name=Default Datasource,process=server1,platform=dynamicproxy,node=
system1Node01,JDBCProvider=Derby JDBC Provider,j2eeType=JDBCDataSource,J2EESe
rver=server1,Server=server1,version=6.0.0.0,type=DataSource,mbeanIdentifier=cell
s/system1Node01Cell/nodes/system1Node01/servers/server1/resources.xml#DataSource
_1094760149902,JDBCResource=Derby JDBC Provider,cell=system1Node01Cell"
```

Now you can use the objectName for getting help on attributes and operations available for this mbean.

```
$Help attributes $objectName
$Help operations $objectName
```

To get or set an attribute or to use an operation, use one of the following commands:

```
$AdminControl getAttribute $objectName "attribute name"
$AdminControl setAttribute $objectName "attribute name" value
$AdminControl invoke $objectName "operation"
```

Attribute, operation examples:

```
Get and set a attribute maxConnections (Note, if you get no value, a null value, or
a java.lang.IllegalStateException, the connection factory or data source for this
MBean has not been created. The connection factory or data source is created at first
JNDI lookup. For this example, the Default Datasource needs to be used before get,
set and invoke will work.)

wsadmin>$AdminControl getAttribute $objectName maxConnections
10

wsadmin>$AdminControl setAttribute $objectName maxConnections 20

wsadmin>$AdminControl getAttribute $objectName maxConnections
20

Using invoke

wsadmin>$AdminControl invoke $objectName showPoolContents

PoolManager name:DefaultDatasource
PoolManager object:746354514
Total number of connections: 3 (max/min 20/1, reap/unused/aged 180/1800/0,
connectiontimeout/purge 1800/EntirePool)
                            (testConnection/inteval false/0, stuck timer/time
/threshold 0/0/0, surge time/connections 0/-1)
Shared Connection information (shared partitions 200)
  No shared connections

Free Connection information (free distribution table/partitions 5/1)
  (2)(0)MCWrapper id 5c6af75b  Managed connection WSRdbManagedConnectionImpl@4b5
a775b  State:STATE_ACTIVE_FREE
```

```
  (2)(0)MCWrapper id 3394375a  Managed connection WSRdbManagedConnectionImpl@328
5f75a  State:STATE_ACTIVE_FREE
  (2)(0)MCWrapper id 4795b75a  Managed connection WSRdbManagedConnectionImpl@46a
4b75a  State:STATE_ACTIVE_FREE

  Total number of connection in free pool: 3
UnShared Connection information
  No unshared connections
```

Here is an example Java program using the wsadmin tool to invoke showPoolContents: the Example: using wsadmin to invoke showPoolContents topic in the *Administering applications and their environment* PDF.

Example: Invoking showPoolContents using the wsadmin tool:

This example Java program uses the wsadmin tool to invoke showPoolContents.

```
/**
 * "This sample program is provided AS IS and may be used, executed, copied and modified without royalty payment
 *  by customer (a) for its own instruction and study, (b) in order to develop applications designed to run with
 *  an IBM WebSphere product, either for customer's own internal use or for redistribution by customer, as part
 *  of such an application, in customer's own products. "

 * Product 5724i63, (C) COPYRIGHT International Business Machines Corp., 2004


 *
 * All Rights Reserved * Licensed Materials - Property of IBM
 *
 * This program will display an active mbean object name and the connection pool.
 *
 * To run this program
 *
 *  1. call "%WAS_HOME%/bin/setupCmdLine.bat"
 *
 *  2. "%JAVA_HOME%\bin\java" "%CLIENTSAS%" "-Dwas.install.root=%WAS_HOME%" "-Dwas.repository.root=%CONFIG_ROOT%"
 *     -Dcom.ibm.CORBA.BootstrapHost=%COMPUTERNAME% -classpath "%WAS_CLASSPATH%;%WAS_HOME%\runtimes
\com.ibm.ws.admin.client_7.0.0.jar;%WAS_HOME%
 *   \lib\wasjmx.jar;." ShowPoolContents DataSource_mbean_name
 */

import java.util.Properties;
import java.util.Set;

import javax.management.*;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.websphere.management.exception.ConnectorException;

public class ShowPoolContents {

  private AdminClient adminClient;

  public static void main(String[] args) {
    try {
      String name2 = null;
      String port = null;
      if (args.length < 2) {
        System.out.println("Enter name for the mbean and port");
        return;
      }
      else {
        name2 = args[0];
        port = args[1];
        System.out.println("Searching for name" + name2);
      }
      ShowPoolContents ace = new ShowPoolContents();

      // Create an AdminClient
      ace.createAdminClient(port);

      ObjectName[] cfON = ace.queryMBean("DataSource", null);

      if (cfON.length == 0) {
        System.out.println(" *Error : queryMBean did not find any active mbeans of type DataSource.");
        System.out.println(" At first touch of a DataSource, an mbean will be created.
To touch a DataSource, use getConnection");
        return;
      }
      int selectedObjectName = -1;
      String findName = name2;
```

```
      for (int i = 0; i < cfON.length; i++) {
        System.out.println("\ncfON[i] = " + cfON[i]);
      }

    }
    catch (Exception e) {
      System.out.println(" *Exception : " + e.toString());
      e.printStackTrace(System.out);
    }
  }

  private void createAdminClient(String port) {
    // Set up a Properties object for the JMX connector attributes
    Properties connectProps = new Properties();
    connectProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
    connectProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
    connectProps.setProperty(AdminClient.CONNECTOR_PORT, port);

    // Get an AdminClient based on the connector properties
    try {
      adminClient = AdminClientFactory.createAdminClient(connectProps);
    }
    catch (ConnectorException e) {
      System.out.println("Exception creating admin client: " + e);
      System.exit(-1);
    }

    System.out.println("Connected to DeploymentManager");
  }

  // helper method to query mbean by type / name
  public ObjectName[] queryMBean(String type, String name) throws Exception {
    String s = "*:";
    if (type != null)
      s += "type=" + type;

    if (name != null)
      s += ",name=" + name;

    s += ",*";

    System.out.println("queryMBean: " + s);
    ObjectName ion = new ObjectName(s);

    Set set = adminClient.queryNames(ion, null);
    Object[] o = set.toArray();
    ObjectName[] on = new ObjectName[o.length];

    for (int i = 0; i < o.length; i++) {
      on[i] = (ObjectName)o[i];
    }

    return on;
  }
}
```

Related tasks

"Changing connection pool settings with the wsadmin tool" on page 1342
You can use the wsadmin scripting tool to change connection pool settings.

Configuring new data source custom properties using scripting

You can configure new data source custom properties using the wsadmin tool and scripting.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new data source custom property:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newds [$AdminConfig getid /Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/]
     ```
   - Using Jython:

```
newds = AdminConfig.getid('/Cell:mycell/Node:mynode/JDBCProvider:JDBC1/DataSource:DS1/')
print newds
```

Example output:

```
DS1(cells/mycell/nodes/mynode|resources.xml$DataSource_1)
```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newds propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newds, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_8)
   ```

3. Get required attribute:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up attributes:
   - Using Jacl:

     ```
     set name [list name RP4]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP4']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new Java 2 Connector authentication data entries using scripting

You can configure new Java 2 Connector (J2C) authentication data entries with the wsadmin tool and scripting.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new J2C authentication data entry:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set security [$AdminConfig getid /Cell:mycell/Security:/]
     ```
   - Using Jython:
     ```
     security = AdminConfig.getid('/Cell:mycell/Security:/')
     print security
     ```

   Example output:
   ```
   (cells/mycell|security.xml#Security_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required JAASAuthData
     ```
   - Using Jython:
     ```
     print AdminConfig.required('JAASAuthData')
     ```

   Example output:
   ```
   Attribute       Type
   alias           String
   userId          String
   password        String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set alias [list alias myAlias]
     set userid [list userId myid]
     set password [list password secret]
     set jaasAttrs [list $alias $userid $password]
     ```

     Example output:
     ```
     {alias myAlias} {userId myid} {password secret}
     ```
   - Using Jython:
     ```
     alias = ['alias', 'myAlias']
     userid = ['userId', 'myid']
     password = ['password', 'secret']
     jaasAttrs = [alias, userid, password]
     print jaasAttrs
     ```

     Example output:
     ```
     [['alias', 'myAlias'], ['userId', 'myid'], ['password', 'secret']]
     ```
4. Create JAAS auth data:
   - Using Jacl:
     ```
     $AdminConfig create JAASAuthData $security $jaasAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('JAASAuthData', security, jaasAttrs)
     ```

   Example output:
   ```
   (cells/mycell|security.xml#JAASAuthData_2)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WAS40 data sources using scripting

Use scripting to configure a new WAS40 data source.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newjdbc [$AdminConfig getid "/JDBCProvider:Apache Derby JDBC Provider/"]
     ```
   - Using Jython:
     ```
     newjdbc = AdminConfig.getid('/JDBCProvider:Apache Derby JDBC Provider/')
     print newjdbc
     ```

   Example output:
   ```
   JDBC1(cells/mycell/nodes/mynode|resources.xml$JDBCProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required WAS40DataSource
     ```
   - Using Jython:
     ```
     print AdminConfig.required('WAS40DataSource')
     ```

   Example output:
   ```
   Attribute    Type
   name    String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name was4DS1]
     set ds4Attrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'was4DS1']
     ds4Attrs = [name]
     ```
4. Create WAS40DataSource:
   - Using Jacl:
     ```
     set new40ds [$AdminConfig create WAS40DataSource $newjdbc $ds4Attrs]
     ```
   - Using Jython:
     ```
     new40ds = AdminConfig.create('WAS40DataSource', newjdbc, ds4Attrs)
     print new40ds
     ```

   Example output:
   ```
   was4DS1(cells/mycell/nodes/mynode|resources.xml#WAS40DataSource_1)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WAS40 connection pools using scripting

You can use scripting to configure a new WAS40 connection pool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WAS40 connection pool:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
     Server:server1/JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynodes:resources.xml$WAS40DataSource_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WAS40ConnectionPool
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WAS40ConnectionPool')
     ```

   Example output:

   ```
   Attribute       Type
   minimumPoolSize     Integer
   maximumPoolSize     Integer
   connectionTimeout   Integer
   idleTimeout        Integer
   orphanTimeout       Integer
   statementCacheSize  Integer
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set mps [list minimumPoolSize 5]
     set minps [list minimumPoolSize 5]
     set maxps [list maximumPoolSize 30]
     set conn [list connectionTimeout 10]
     set idle [list idleTimeout 5]
     set orphan [list orphanTimeout 5]
     set scs [list statementCacheSize 5]
     set 40cpAttrs [list $minps $maxps $conn $idle $orphan $scs]
     ```

     Example output:

     ```
     {minimumPoolSize 5} {maximumPoolSize 30}
     {connectionTimeout 10} {idleTimeout 5}
     {orphanTimeout 5} {statementCacheSize 5}
     ```

   - Using Jython:

     ```
     minps = ['minimumPoolSize', 5]
     maxps = ['maximumPoolSize', 30]
     conn = ['connectionTimeout', 10]
     idle = ['idleTimeout', 5]
     orphan = ['orphanTimeout', 5]
     scs = ['statementCacheSize', 5]
     cpAttrs = [minps, maxps, conn, idle, orphan, scs]
     print cpAttrs
     ```

     Example output:

     ```
     [[minimumPoolSize, 5], [maximumPoolSize, 30],
     [connectionTimeout, 10], [idleTimeout, 5],
     [orphanTimeout, 5], [statementCacheSize, 5]]
     ```

4. Create was40 connection pool:
   - Using Jacl:

```
$AdminConfig create WAS40ConnectionPool $new40ds $40cpAttrs
```

- Using Jython:

```
print AdminConfig.create('WAS40ConnectionPool', new40ds, 40cpAttrs)
```

Example output:

```
(cells/mycell/nodes/mynode:resources.xml#WAS40ConnectionPool_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WAS40 custom properties using scripting

You can use scripting and the wsadmin tool to configure a new WAS40 custom property.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WAS40 custom properties:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set new40ds [$AdminConfig getid /Cell:mycell/Node:mynode/
     JDBCProvider:JDBC1/WAS40DataSource:was4DS1/]
     ```

   - Using Jython:

     ```
     new40ds = AdminConfig.getid('/Cell:mycell/Node:mynode/
     JDBCProvider:JDBC1/WAS40DataSource:was4DS1/')
     print new40ds
     ```

   Example output:

   ```
   was4DS1(cells/mycell/nodes/mynodes|resources.xml$WAS40DataSource_1)
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newds propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newds, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_9)
   ```

3. Get required attribute:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute    Type
   name         String
   ```

4. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name RP5]
     set rpAttrs [list $name]
     ```

   - Using Jython:

```
name = ['name', 'RP5']
rpAttrs = [name]
```

5. Create J2EE Resource Property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP5(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_9)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new J2C resource adapters using scripting

Use the wsadmin tool to configure resource adapters with Resource Adapter Archive (RAR) files. A RAR file provides the classes and other code to support a resource adapter for access to a specific enterprise information system (EIS), such as the Customer Information Control System (CICS). Configure resource adapters for an EIS only after you install the appropriate RAR file.

Before you begin

A RAR file, which is often called a J2EE Connector Architecture (JCA) connector, must comply with the JCA Specification. Meet these requirements by using a supported assembly tool (as described in the Assembly tools article) to assemble a collection of Java archive (JAR) files, other runnable components, utility classes, and so on, into a deployable RAR file. Then you are ready to install your RAR file in Application Server.

There are two ways to complete this task. This topic uses the AdminConfig object to install resource adapters. Alternatively, you can use the installJ2CResourceAdapter script in the AdminJ2C script library to install a J2C resource adapter in your configuration, as the following example demonstrates:

```
AdminJ2C.installJ2CResourceAdapter("myNode", "C:\temp\jca15cmd.rar", "J2CTest")
```

The scripting library provides a set of procedures to automate the most common administration functions. You can run each script procedure individually, or combine several procedures to quickly develop new scripts.

1. Launch the wsadmin scripting tool using the Jython scripting language.
2. Identify the configuration ID of the node to which the resource adapter is installed, as the following examples demonstrate:
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

3. Identify the optional attributes.

   The J2CResourceAdapter object does not require specific arguments. Use the following command to display the optional attributes for the J2CResourceAdapter object:
   - Using Jacl:

     ```
     $AdminConfig defaults J2CResourceAdapter
     ```

   - Using Jython:

     ```
     print AdminConfig.defaults('J2CResourceAdapter')
     ```

The following displays the command output that displays each optional attribute and the data type for the attribute, and denotes the default attributes:

```
Attribute                       Type
name                            String
description                     String
classpath                       String
nativepath                      String
providerType                    String
archivePath                     String
threadPoolAlias                 String
propertySet                     J2EEResourcePropertySet
jaasLoginConfiguration          JAASConfigurationEntry
deploymentDescriptor            Connector
connectionDefTemplateProps      ConnectionDefTemplateProps
activationSpecTemplateProps     ActivationSpecTemplateProps
j2cAdminObjects                 J2CAdminObject
adminObjectTemplateProps        AdminObjectTemplateProps
j2cActivationSpec               J2CActivationSpec
```

4. Set up the attributes of interest.

   Determine the attributes to configure for the J2C resource adapter. In the following examples, the commands set the RAR file path to the `rarFile` variable and the name and description configuration options to the `option` variable:

   - Using Jacl:

     ```
     set rarFile /currentScript/cicseci.rar
     set option {-rar.name  RAR1 -rar.desc "New resource adapter"}
     ```

   - Using Jython:

5. Create a resource adapter.

   Use the installResourceAdapter command for the AdminConfig object to install the resource adapter with the previously set configuration options, as the following examples demonstrate:

   - Using Jacl:

     ```
     $AdminConfig installResourceAdapter $rarFile mynode $option
     ```

   - Using Jython:

     ```
     AdminConfig.installResourceAdapter(rarFile, 'mynode', option)
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

6. Save the configuration changes.

Configuring custom properties for J2C resource adapters using scripting

You can configure custom properties for J2C resource adapters with scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a J2C resource adapters:

1. Identify the parent ID and assign it to the newra variable.

   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```

   - Using Jython:

```
newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
print newra
```

Example output:

```
RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newra propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newra, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_8)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name          String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP4]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP4']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP4(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_8)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new J2C connection factories using scripting

Use scripting and the wsadmin tool to configure new J2C connection factories.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new J2C connection factory:

1. Identify the parent ID and assign it to the newra variable.
   - Using Jacl:

```
set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
```

- Using Jython:

```
newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
print newra
```

Example output:

```
RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
```

2. There are two ways to configure a new J2C connection factory. Perform one of the following:
   - Using the AdminTask object:
     a. List the connection factory interfaces:
        - Using Jacl:

          ```
          $AdminTask listConnectionFactoryInterfaces $newra
          ```

        - Using Jython:

          ```
          AdminTask.listConnectionFactoryInterfaces(newra)
          ```

          Example output:

          ```
          javax.sql.DataSource
          ```

     b. Create a J2CConnectionFactory:
        - Using Jacl:

          ```
          $AdminTask createJ2CConnectionFactory $newra { -name cf1
          -jndiName eis/cf1 -connectionFactoryInterface
          avax.sql.DataSource
          ```

        - Using Jython:

          ```
          AdminTask.createJ2CConnectionFactory(newra, '['-name', 'cf1',
          '-jndiName', 'eis/cf1', '-connectionFactoryInterface',
          'avax.sql.DataSource']')
          ```

   - Using the AdminConfig object:
     a. Identify the required attributes:
        - Using Jacl:

          ```
          $AdminConfig required J2CConnectionFactory
          ```

        - Using Jython:

          ```
          print AdminConfig.required('J2CConnectionFactory')
          ```

          Example output:

          ```
          Attribute Type
          connectionDefinition ConnectionDefinition@
          ```

     b. If your resource adapter is JCA1.5 and you have multiple connection definitions defined, it is required that you specify the ConnectionDefinition attribute. If your resource adapter is JCA1.5 and you have only one connection definition defined, it will be picked up automatically. If your resource adapter is JCA1.0, you do not need to specify the ConnectionDefinition attribute. Perform the following command to list the connection definitions defined by the resource adapter:
        - Using Jacl:

          ```
          $AdminConfig list ConnectionDefinition $newra
          ```

        - Using Jython:

          ```
          print AdminConfig.list('ConnectionDefinition', $newra)
          ```

     c. Set up the required attributes:
        - Using Jacl:

          ```
          set name [list name J2CCF1]
          set jname [list jndiName eis/j2ccf1]
          set j2ccfAttrs [list $name]
          ```

        - Using Jython:

```
name = ['name', 'J2CCF1']
jname = ['jndiName', 'eis/j2ccf1']
j2ccfAttrs = [name,jname]
```

d. If you are specifying the ConnectionDefinition attribute, also set up the following:
   - Using Jacl:
     ```
     set cdattr [list connectionDefinition $cd]
     ```
   - Using Jython:
     ```
     cdattr = ['connectionDefinition', $cd]
     ```

e. Create a J2C connection factory:
   - Using Jacl:
     ```
     $AdminConfig create J2CConnectionFactory $newra $j2ccfAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('J2CConnectionFactory', newra, j2ccfAttrs)
     ```

   Example output:
   ```
   J2CCF1(cells/mycell/nodes/mynode|resources.xml#J2CConnectionFactory_1)
   ```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new J2C activation specifications using scripting

You can configure new J2C activation specifications using scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a J2C activation specifications:

1. Identify the parent ID and assign it to the `newra` variable.
   - Using Jacl:
     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```
   - Using Jython:
     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:
   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```

2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:
        ```
        $AdminTask listMessageListenerTypes $newra
        ```
        Using Jython:
        ```
        AdminTask.listMessageListenerTypes(newra)
        ```
        Example output:
        ```
        javax.jms.MessageListener
        ```
     b. Create a J2C administrative object:

        Using Jacl:

```
$AdminTask createJ2CActivationSpec $newra { -name ac1
-jndiName eis/ac1 -messageListenerType
javax.jms.MessageListener}
```

Using Jython:

```
AdminTask.createJ2CActivationSpec(newra, ['-name', 'ao1',
'-jndiName', 'eis/ao1', '-messageListenerType',
'javax.jms.MessageListener'])
```

- Using the AdminConfig object:
  a. Using Jacl:

```
$AdminConfig required J2CActivationSpec
```

Using Jython:

```
print AdminConfig.required('J2CActivationSpec')
```

Example output:

```
Attribute Type
activationSpec ActivationSpec@
```

  b. If your resource adapter is JCA V1.5 and you have multiple activation specifications defined, it is required that you specify the activation specification attribute. If your resource adapter is JCA V1.5 and you have only one activation specification defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the activationSpec attribute. Perform the following command to list the activation specifications defined by the resource adapter:

Using Jacl:

```
$AdminConfig list ActivationSpec $newra
```

Using Jython:

```
print AdminConfig.list('ActivationSpec', $newra)
```

  c. Set the administrative object that you need to a variable:

Using Jacl:

```
set ac [$AdminConfig list ActivationSpec $newra]
set name [list name J2CAC1]
set jname [list jndiName eis/J2CAC1]
set j2cacAttrs [list $name $jname $cdcttr]
```

Using Jython:

```
ac = AdminConfig.list('ActivationSpec', $newra)
name = ['name', 'J2CAC1']
jname = ['jndiName', 'eis/j2cac1']
j2cacAttrs = [name, jname,cdattr]
```

  d. If you are specifying the ActivationSpec attribute, also set up the following:

Using Jacl:

```
set cdcttr [list activationSpec $ac]
```

Using Jython:

```
cdattr = ['activationSpec', ac]
```

  e. Create a J2C activation specification object:

Using Jacl:

```
$AdminConfig create J2CActivationSpec $newra $j2cacAttrs
```

Using Jython:

```
print AdminConfig.create('J2CActivationSpec', newra,j2cacAttrs)
```

Example output:

```
J2CAC1(cells/mycell/nodes/mynode|resources.xml#J2CActivationSpec_1)
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new J2C administrative objects using scripting

You can use scripting and the wsadmin tool to configure new J2C administrative objects.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a J2C administrative object:

1. Identify the parent ID and assign it to the `newra` variable.
   - Using Jacl:

     ```
     set newra [$AdminConfig getid /Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/]
     ```
   - Using Jython:

     ```
     newra = AdminConfig.getid('/Cell:mycell/Node:mynode/J2CResourceAdapter:RAR1/')
     print newra
     ```

   Example output:

   ```
   RAR1(cells/mycell/nodes/mynode|resources.xml#J2CResourceAdapter_1)
   ```
2. There are two ways to configure a new J2C administrative object. Perform one of the following:
   - Using the AdminTask object:
     a. List the administrative object interfaces:

        Using Jacl:

        ```
        $AdminTask listAdminObjectInterfaces $newra
        ```

        Using Jython:

        ```
        AdminTask.listAdminObjectInterfaces(newra)
        ```

        Example output:

        ```
        com.ibm.test.message.FVTMessageProvider
        ```

     b. Create a J2C administrative object:

        Using Jacl:

        ```
        $AdminTask createJ2CAdminObject $newra { -name ao1 -jndiName eis/ao1
        -adminObjectInterface com.ibm.test.message.FVTMessageProvider }
        ```

        Using Jython:

        ```
        AdminTask.createJ2CAdminObject(newra, ['-name', 'ao1', '-jndiName', 'eis/ao1',
        '-adminObjectInterface', 'com.ibm.test.message.FVTMessageProvider'])
        ```
   - Using the AdminConfig object:
     a. Using Jacl:

        ```
        $AdminConfig required J2CAdminObject
        ```

        Using Jython:

        ```
        print AdminConfig.required('J2CAdminObject')
        ```

        Example output:

        ```
        Attribute Type
        adminObject AdminObject@
        ```

     b. If your resource adapter is JCA V1.5 and you have multiple administrative objects defined, it is required that you specify the administrative object attribute. If your resource adapter is JCA V1.5 and you have only one administrative object defined, it will be picked up automatically. If your resource adapter is JCA V1.0, you do not need to specify the administrative object attribute. Perform the following command to list the administrative objects defined by the resource adapter:

        Using Jacl:

```
$AdminConfig list AdminObject $newra
```

Using Jython:

```
print AdminConfig.list('AdminObject', $newra)
```

c. Set the administrative objects that you need to a variable:

Using Jacl:

```
set ao AdminObjectId
set name [list name J2CAO1]
set jname [jndiName eis/j2cao1]
set j2caoAttrs [list $name $jname]
```

Using Jython:

```
ao = AdminObjectId
name = ['name', 'J2CAO1']
set jname = ['jndiName', eis/j2cao1]
j2caoAttrs = [name, jname]
```

d. If you are specifying the AdminObject attribute, also set up the following:

Using Jacl:

```
set cdattr [list adminObject $ao]
```

Using Jython:

```
cdattr = ['adminObject', ao]
```

e. Create a J2C administrative object:

Using Jacl:

```
$AdminConfig create J2CAdminObject $newra $j2caoAttrs
```

Using Jython:

```
print AdminConfig.create('J2CAdminObject', newra, j2caoAttrs)
```

Example output:

```
J2CAO1(cells/mycell/nodes/mynode|resources.xml#J2CAdminObject_1)
```

3. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Managing the message endpoint lifecycle using scripting

Use the Jython scripting language to manage your message endpoints with the wsadmin tool. Use this topic to query your configuration for message endpoint properties, and to deactivate or reactivate a message endpoint.

About this task

The Java EE Connector Architecture (JCA) allows the application server to link inbound requests from messaging resource adapters to message endpoints. The message endpoint managed bean (MBean) is a Java Management Extensions (JMX) framework MBean that the application server associates with a message endpoint instance.

Use this topic to manage situations where messaging providers fail to deliver messages to their intended destinations. For example, a provider might fail to deliver messages to a message endpoint when its underlying Message Driven Bean attempts to commit transactions against a database server that is not responding. To troubleshoot the problem, use the wsadmin tool to temporarily disable the message endpoint from handling messages. After troubleshooting the issue, use the wsadmin tool to reactivate the message endpoint.

The steps in this topic display how to use the AdminControl object and the wsadmin tool to invoke a Message Endpoint MBean to:

- Display properties of the a message endpoint
- Temporarily deactivate a message endpoint

- Reactivate a message endpoint

Note: Starting in Version 7.0, you can use the AdminControl object and the wsadmin tool to deactivate message endpoints to pause the endpoints from receiving messages, and to reactivate message endpoints to resume message handling. Previously, the application server activated and deactivated message endpoints when the application or resource adapter was started and stopped.

- Display properties of a message endpoint.

  Use the following command to display a list of all message endpoints in your configuration:

```
AdminControl.queryNames('*:type=J2CMessageEndpoint,*')
```

  For the previous example, the command returns the following information for the `JMSMDB_MessageEndpoint`:

```
WebSphere:name=JMSMDB_J2CMessageEndpoint,
process=myServer,
platform=dynamicproxy,
cell=myNode01Cell,
node=myNode01,
version=6.1.0.0,
type=J2CMessageEndpoint,
mbeanIdentifier=cells/myNode01Cell/nodes/myNode01/servers/myServer/resources.xml#example#example.jar#JMSMDB_J2CMessageEndpoint,
spec=1.0,
Server=server1,
diagnosticProvider=false,
j2eeType=JCAMessageEndpoint,
J2EEApplication=example
J2EEServer=myServer,
J2CResourceAdapter=SIB JMS Resource Adapter,
MessageDrivenBean=example#example.jar#JMSMDBActivationSpec=3727AS1
```

- Temporarily deactivate the message endpoint.

  Use the following commands to cache the instance and deactivate the message endpoint:

```
objectName=AdminControl.queryNames('*:name=JMSMDB_MessageEndpoint,*')
AdminControl.invoke(objectName, 'pause')
```

  The command returns the following output:

```
Message Endpoint JMSMDB_J2CMessageEndpoint is deactivated
```

- Reactivate the message endpoint.

  Use the following commands to cache the instance and reactivate the message endpoint:

```
objectName=AdminControl.queryNames('*:name=JMSMDB_MessageEndpoint,*')
AdminControl.invoke(objectName, 'resume')
```

  The command returns the following output:

```
Message Endpoint JMSMDB_J2CMessageEndpoint is activated
```

Testing data source connections using scripting

You can test connections for data sources with the wsadmin tool and scripting. After you have defined and saved a data source, you can test the data source connection to ensure that the parameters in the data source definition are correct.

About this task

You can use the testConnection command for the AdminControl object to test data source connections for a cell, node, server, application, or cluster scope. This topic provides an example that tests the data source connection for the application scope.

- Test the data source connection for a cell, node, or server scope.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Identify the DataSourceCfgHelper MBean and assign it to the dshelper variable.
     - Using Jacl:

```
set ds [$AdminConfig getid /DataSource:DS1/]
$AdminControl testConnection $ds
```

– Using Jython:

```
ds = AdminConfig.getid('/DataSource:DS1/')
AdminControl.testConnection(ds)
```

Example output:

```
WASX7217I: Connection to provided datasource was successful.
```

3. Test the connection. The following example invokes the testConnectionToDataSource operation on the MBean, passing in the classname, userid, password, database name, JDBC driver class path, language, and country.
   – Using Jacl:
   – Using Jython:

   Example output:

```
WASX7217I: Connection to provided data source was successful.
```

- Test the data source connection for an application scope.
  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Get the data source for the application of interest.

     Use the AdminConfig object to determine the configuration IDs of the `myApplication` application and `DSA1` data source, as the following examples demonstrate:

     – Using Jacl:

```
set appID [$AdminConfig getid /Deployment:myApplication/]
set ds [$AdminConfig list DataSource $appID]
```

     – Using Jython:

```
appID = AdminConfig.getid("/Deployment:myApplication/")
ds = AdminConfig.list("DataSource", appID)
```

  3. Test the connection.

     Use the AdminConfig object to test the connection for the data source of interest, as the following examples demonstrate:

     – Using Jacl:

```
$AdminControl testConnection $ds
```

     – Using Jython:

```
AdminControl.testConnection(ds)
```

  The command returns output that indicates whether the connection is successful, as demonstrated in the following sample output:

```
WASX7467I: Connection to provided datasource on node myNode processnodeagent was successful.
WASX7217I: Connection to provided datasource was successful.
```

- Test the data source connection for a cluster scope.

  In the following example, the `Cluster1` server cluster contains cluster members on the `node1`, `node2`, and `node3` nodes. The `Cluster1` server cluster contains the `DSC1` data source.

  1. Launch the wsadmin scripting tool using the Jython scripting language.
  2. Get the data source configuration ID for the cluster of interest.

     Use the AdminConfig object to determine the configuration IDs of the `Cluster1` cluster and `DSA1` data source, as the following examples demonstrate:

     – Using Jacl:

```
set cluster [$AdminConfig getid /ServerCluster:Cluster1/]
set ds [$AdminConfig list DataSource $cluster]
```

     – Using Jython:

```
cluster = AdminConfig.getid("/ServerCluster:Cluster1/")
ds = AdminConfig.list("DataSource", cluster)
```

  3. Test the connection.

Use the AdminConfig object to test the connection for the data source of interest, as the following examples demonstrate:

– Using Jacl:

```
$AdminControl testConnection $ds
```

– Using Jython:

```
AdminControl.testConnection(ds)
```

The command returns output that indicates whether the connection is successful, as demonstrated in the following sample output:

```
WASX7467I: Connection to provided datasource on node node1 process nodeagent was successful.
WASX7467I: Connection to provided datasource on node node2 process nodeagent was successful.
WASX7467I: Connection to provided datasource on node node3 process nodeagent was successful.
WASX7217I: Connection to provided datasource was successful.
```

JDBCProviderManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages in interactive mode to configure data access and data sources with scripting. The commands and parameters in the JDBCProviderManagement group can be used to create or list data sources and Java database connectivity (JDBC) providers.

The JDBCProviderManagement command group for the AdminTask object includes the following commands:

- "createDatasource"
- "createJDBCProvider" on page 1365
- "listDatasources" on page 1367
- "listJDBCProviders" on page 1367

createDatasource

The createDatasource command creates a new data source to access the back end data store. Application components use the data source to access connection instances to your database.

Target object

JDBC Provider Object ID - The configuration object of the JDBC provider to which the new data source will belong.

Required parameters

- name
The name of the data source. (String, required)

-jndiName
The Java Naming and Directory Interface (JNDI) name. (String, required)

-dataStoreHelperClassName
The name of the DataStoreHelper implementation class that extends the capabilities of the selected JDBC driver implementation class to perform data-specific functions. WebSphere Application Server provides a set of DataStoreHelper implementation classes for each of the JDBC provider drivers it supports. (String, required)

configureResourceProperties
This command step configures the resource properties that are required by the data source. (Required) You can specify the following parameters for this step:

name
The name of the resource property. (String, required)

type
The type of the resource property. (String, required)

value
The value of the resource property. (String, required)

Optional parameters

-description
The description of the data source. (String, optional)

-category
The category that you can use to classify a group of data sources. (String, optional)

-componentManagedAuthenticationAlias
The alias used for database authentication at run time. This alias is only used when the application resource reference is using res-auth=Application. (String, optional)

-containerManagedPersistence
Specifies if the data source is used for container managed persistence for enterprise beans. The default value is true. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createDatasource "DB2 Universal JDBC Driver Provider (XA)(cells/myCell|resources.xml#JDBCProvider_1180538152781)" {-name
"DB2  Universal JDBC Driver XA DataSource" -jndiName s1 -dataStoreHelperClassName com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
 -componentManagedAuthenticationAlias myCellManager01/a1 -xaRecoveryAuthAlias myCellManager01/a1 -configureResourceProperties
{{databaseName java.lang.String db1} {driverType java.lang.Integer 4} {serverName java.lang.String  dbserver1} {portNumber java.lang.Integer 50000}}}
```

- Using Jython string:

```
AdminTask.createDatasource('DB2 Universal JDBC Driver Provider(XA) (cells/myCell|resources.xml#JDBCProvider_1180501752515)', '[-name
"DB2 Universal JDBC Driver XA DataSource 2" -jndiName ds2 -dataStoreHelperClassName com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper
 -componentManagedAuthenticationAlias myCellManager01/a1 -xaRecoveryAuthAlias myCellManager01/a1 -configureResourceProperties
[[databaseName java.lang.String db1] [driverType java.lang.Integer 4] [serverName java.lang.String  dbserver1] [portNumber java.lang.Integer 50000]]]')
```

- Using Jython list:

```
AdminTask.createDatasource('DB2 Universal JDBC Driver Provider(XA) (cells/myCell|resources.xml#JDBCProvider_1180501752515)', ['-name', '
DB2  Universal JDBC Driver XA DataSource 2', '-jndiName', 'ds2', '-dataStoreHelperClassName',
 'com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper', '-componentManagedAuthenticationAlias', 'myCellManager01/a1',
 '-xaRecoveryAuthAlias', 'myCellManager01/a1', '-configureResourceProperties', '[[databaseName java.lang.String db1]
[driverType java.lang.Integer 4] [serverName java.lang.String  dbserver1] [portNumber java.lang.Integer 50000]]']
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createDatasource {-interactive}
```

- Using Jython:

```
AdminTask.createDatasource('-interactive')
```

createJDBCProvider

The createJDBCProvider command creates a new Java database connectivity provider (JDBC) that you can use to connect to a relational database for data access.

Target object

None

Required parameters

-scope
The scope for the new JDBC provider. Provide the scope in the form `type=name`. `Type` can be Cell,

Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `none`. (String, required)

-databaseType
The type of database that will be used by the JDBC provider. For example, DB2, Derby, Informix, Oracle, and so on. (String, required)

-providerType
The JDBC provider type that will be used by the JDBC provider. (String, required)

-implementationType
The implementation type for this JDBC provider. Use `Connection pool data source` if your application runs in a single phase or a local transaction. Otherwise, use `XA data source` to run in a global transaction. (String, required)

Optional parameters

-classpath
Specifies a list of paths or JAR file names that form the location for the resource provider classes. (String, optional)

-description
The description for the JDBC provider. (String, optional)

-implementationClassName
Specifies the Java class name for the JDBC driver implementation. (String, optional)

-isolated
Specifies that the JDBC provider will load in its own class loader. The default value is false. You cannot specify a native path for an isolated JDBC provider. (Boolean, optional)

-name
The name of the JDBC provider. The default is the value from the provider template. (String, optional)

-nativePath
Specifies a list of paths that form the location for the resource provider native libraries. (String, optional)

-isolated
Specifies whether the JDBC provider loads within the class loader. The default value is `false`. You cannot specify a native path for an isolated JDBC provider. (Boolean, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJDBCProvider {-scope Cell=my02Cell -databaseType DB2 -providerType "DB2 Universal JDBC Driver Provider"
-implementationType "XA data source" -name "DB2 Universal JDBC Driver Provider (XA)" -description "XA DB2 Universal JDBC Driver-compliant
Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on WAS z/OS is not
supported for datasources created under this provider." -classpath
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar -nativePath ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}}
```

- Using Jython string:

```
AdminTask.createJDBCProvider('[-scope Cell=myCell -databaseType DB2 -providerType "DB2 Universal JDBC Driver Provider"
 -implementationType "XA datasource" -name "DB2 Universal JDBC Driver Provider (XA)" -description "XA DB2 Universal JDBC
Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver
 type 2 on WAS z/OS is not supported for datasources created under this provider." -classpath
 ${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar -nativePath ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}]')
```

- Using Jython list:

```
AdminTask.createJDBCProvider(['-scope', 'Cell=myCell', '-databaseType', 'DB2', '-providerType', 'DB2 Universal JDBC Driver Provider',
 '-implementationType', 'XA data source', '-name', 'DB2 Universal JDBC Driver Provider (XA)', '-description', 'XA DB2 Universal
JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of
driver type 2 on WAS z/OS is not supported for datasources created under this provider.', '-classpath',
  '${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar;${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar;${DB2UNIVERSAL_JDBC_DRIVER_PATH}/
db2jcc_license_cisuz.jar', '-nativePath', '${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJDBCProvider {-interactive}
```

- Using Jython:

```
AdminTask.createJDBCProvider('-interactive')
```

listDatasources

Use the listDatasources command to list data sources that are contained in the specified scope.

Target object

None

Required parameters

None.

Optional parameters

-scope
The scope for the data sources that will be listed. Provide the scope in the form `type=name`. `Type` can be Cell, Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `All`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listDatasources {-scope Cell=my02Cell}
```

- Using Jython string:

```
AdminTask.listDatasources('[-scope Cell=my02Cell]')
```

- Using Jython list:

```
AdminTask.listDatasources('-scope', 'Cell=my02Cell')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listDatasources {-interactive}
```

- Using Jython:

```
AdminTask.listDatasources('-interactive')
```

listJDBCProviders

The listJDBCProviders command lists JDBC providers that are contained in the specified scope.

Target object

None

Required parameters

None.

Optional parameters

-scope

The scope for the JDBC providers that will be listed. Provide the scope in the form `type=name`. `Type` can be Cell, Node, Server, Application, or Cluster, and `name` is the name of the specific instance of the cell, node, server, application, or cluster that you are using. The default is `All`. (String, optional)

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJDBCProviders {-scope Cell=my02Cell}
```

- Using Jython string:

```
AdminTask.listJDBCProviders('-scope Cell=my02Cell')
```

- Using Jython list:

```
AdminTask.listJDBCProviders(['-scope', 'Cell=my02Cell'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJDBCProviders {-interactive}
```

- Using Jython:

```
AdminTask.listJDBCProviders('-interactive')
```

Related tasks

Using the AdminTask object for scripted administration
Use the AdminTask object to access a set of administrative commands that provide an alternative way to access the configuration commands and the running object management commands.

Related reference

Commands for the AdminTask object
Use the AdminTask object to run administrative commands with the wsadmin tool.

Configuring Persistence Provider support in the application server

Persistence providers are implementations of the Java Persistence API (JPA) specification and can be deployed in the Java EE compliant application server that supports JPA persistence.

About this task

The Enterprise JavaBeans (EJB) 3.0 specifications require that an application server container that supports the EJB 3.0 programming model must provide a JPA implementation. This is also referred to as a persistence provider. There are two built-in JPA persistence providers: the JPA persistence provider for the application server and the OpenJPA persistence provider. If an explicit provider element is not specified in the persistence unit definitions, the application server will use the default persistence provider, which is the JPA persistence provider for the application server.

The application server provides both the default persistence provider and the Apache OpenJPA persistence provider to support the open source implementation of JPA and allow for easy migration of existing OpenJPA applications to the application server's solution for JPA.

- **Configuring a default persistence provider**

  Use the default persistence provider, or specify a persistence provider for your application. With the application server's JPA persistence provider, you can take advantage of the stability and application extensions that are provided with the application server's implementation of JPA. You can decide which persistence implementation best fits your needs. If an application relies on a specific persistence provider for certain functions and settings, you should specify the provider in the persistence unit definition to avoid any incompatibilities.
- **Using third-party persistence providers**

Use third-party persistence providers in the application server environment.

Configuring the Java Persistence API (JPA) default persistence provider

Two persistence providers are included in WebSphere Application Server: *JPA for WebSphere Application Server persistence provider* and *Apache OpenJPA persistence provider*. The JPA for WebSphere Application Server persistence provider is the default provider for WebSphere Application Server. You can use one of the two providers, or a third-party persistence provider, as the default provider.

About this task

On the application server's feature pack for enterprise beans (EJB) 3.0 the default persistence provider, default Java Transaction API (JTA) datasource Java Naming and Directory Interface (JNDI) name, and default non-JTA datasource JNDI name, values are set through the following Java virtual machine (JVM) properties: com.ibm.websphere.jpa.default.provider, com.ibm.websphere.jpa.default.jta.datasource, and com.ibm.websphere.jpa.default.nonjta.datasource. Support for these properties has been deprecated. Any values that were set through these properties will be displayed as default values on this panel. Any change to these values through the Administrative Console panel will override any values set via the JVM properties.

While built from the Apache OpenJPA persistence provider, the JPA for WebSphere Application Server persistence provider contains the following enhancements and differences:

- Enhanced tracing support
- Version ID generation
- ObjectGrid cache plug-in support
- WebSphere product-specific commands and scripts
- Translated message files
- Static SQL support using the DB2 pureQuery feature
- Access intent support
- The following table shows how the default values for the JPA for WebSphere Application Server provider configuration properties are different from the Apache OpenJPA provider:

Property	**Apache OpenJPA default value**	**JPA for WebSphere Application Server default value**
openjpa.Compatibility	StrictIdentityValues=false	StrictIdentityValues=true
openjpa.RuntimeUnenhancedClasses	supported	warn

If no JPA provider is configured in the <provider> element of the `persistence.xml` file within an EJB module, the default JPA provider that is currently configured for this server is used. The WebSphere Application Server is packaged with the JPA for WebSphere Application Server persistence provider defined as the default provider. However, it is possible to override this default and specify a different default via the Websphere Administrative Console.

You can set your default persistence provider in one of two ways. Use one of the following procedures.

1. Select the default JPA provider from a list of providers included with WebSphere Application Server.
   a. Open the administrative console.
   b. From the left-hand navigation, navigate to **Servers** → **Application Servers**
   c. Select a *server*.
   d. Navigate to **Container Services** → **Default Java Persistence API Settings**

e. Select **Select a default persistence provider that is included with WebSphere Application Server**.
f. Click the slider and select from the list.
g. Click **Apply** and save the configuration.

2. Specify an alternative default persistence provider via the Administrative Console
   a. Open the administrative console.
   b. From the left-hand navigation, navigate to **Servers** → **Application Servers**
   c. Select a *server*.
   d. Navigate to **Container Services** → **Default Java Persistence API Settings**
   e. Select **Specify an alternative default persistence provider** .
   f. Enter the fully qualified JPA implementation class name of a JPA persistence provider in the box.
   g. Click **Apply** and save the configuration.

Related tasks

"Configuring Persistence Provider support in the application server" on page 1368
Persistence providers are implementations of the Java Persistence API (JPA) specification and can be deployed in the Java EE compliant application server that supports JPA persistence.

"Using third-party persistence providers" on page 1371
Java Persistence API (JPA) for WebSphere Application Server allows users to specify third-party persistence providers in their application server environment.

Default Java Persistence API settings

The Java Persistence API (JPA) specification requires a default provider to be defined. If you have applications that use JPA, it is recommended you use this page to provide default values. To increase the portability of your applications, you can use this page to configure the default JPA settings for applications running on this server instead of defining the <provider> element in each persistence unit in your applications. The JPA settings defined here are used for the persistence unit of an application only when the application does not define the JPA settings for that persistence unit.

Note: Application JPA settings always override the settings on this page.

To view this administrative console page, click **Applications** → **server** → **Container Services** → **Default Java Persistence API settings**.

Default persistence provider:

Specify the default persistence provider for the application server container. The default persistence provider can be selected either from a list of providers that are included with the product or a user-specified alternate persistence provider.

Select a persistence provider from the product list or specify a fully package qualified JPA implementation class name of an alternate persistence provider.

Default

com.ibm.websphere.persistence.PersistenceProviderImpl

Note: If an alternate persistence provider is specified as the default, make sure the alternate persistence provider is created in the server. See the information center topic on using a third-party persistence provider.

Default Java Transaction API (JTA) data source Java Naming and Directory Interface (JNDI) name:

Specify the default JTA data source used by persistence units for the application server container.

Select the JNDI name for the data source from the drop down box. The JTA data sources that are currently configured and visible to the application server are available in the drop down box selection.

Default
None

Note: If a default JTA data source is not specified, ensure an appropriate JTA data source is specified in the <jta-data-source> or connection properties field in the <properties> element in the persistence unit.

Default non-JTA data source JNDI name:

Specify the default non-JTA data source used by persistence units for the application server container.

Select the JNDI name for the data source from the drop down box. The data sources that are currently configured, visible to the application server, and are set to ″non-transactional″ are available in the drop down box selection.

Default
None

Note: Some JPA entity features will require a non-JTA data source to be specified. An example of this is automatic entity identity generation. Ensure a non-JTA data source is configured to match your application needs. For information on configuring a non-JTA data source, see the information center topic on associating persistence units and data sources.

Configuring the default JTA and non-JTA data source JNDI Names

The Java Transaction API (JTA) and non-JTA data sources to be used for an application can be specified through the <jta-data-source> and <non-jta-data-source> elements of the `persistence.xml` file within an Enterprise JavaBeans (EJB) module.

About this task

If these elements are not configured in the persistence.xml file, then the default JTA and non-JTA data sources configured for the server are used. These values are null by default. However, the default JTA and non-JTA data source Java Naming and Directory Interface (JNDI) names to be used by applications running on this server can be defined using the administrative console.

1. Open the administrative console.
2. Select **Servers** → **Application server** → ***server*** → **Container Services**.
3. Click **Default Java Persistence API settings**.
4. Select the **Default JTA data source JNDI name** or the **Default non-JTA data source JNDI name** field.
5. Click **Apply** to save the configuration.

Related tasks

“Configuring the Java Persistence API (JPA) default persistence provider” on page 1369
Two persistence providers are included in WebSphere Application Server: *JPA for WebSphere Application Server persistence provider* and *Apache OpenJPA persistence provider*. The JPA for WebSphere Application Server persistence provider is the default provider for WebSphere Application Server. You can use one of the two providers, or a third-party persistence provider, as the default provider.

Using third-party persistence providers

Java Persistence API (JPA) for WebSphere Application Server allows users to specify third-party persistence providers in their application server environment.

About this task

Java EE applications that use JPA functions can employ third-party persistence providers other than those that are included with the application server. Applications can also specify an Apache OpenJPA provider that is other than the version that is included with the application server. There are two basic means to incorporate third-party providers into an application:

- Embedding the persistence provider inside an application
- Making use of shared libraries.

Depending on your requirements, you can embed a persistence provider inside an application, or place the persistence provider into a shared library.

- **Embedding a third-party persistence provider within an application** Sometimes an application design must rely on the implementation of a specific persistence provider. The JPA specification permits embedding of a specific JPA provider within a persistence archive. When building the application, you can assemble a specific version of a provider's implementation in the application enterprise archive (EAR) file or in a web application (WAR) file. To embed a third-party persistence provider into an application, you need to inspect the application design and all dependent prerequisites. Use the following steps to embed a persistence provider inside an application:
  1. Modify the <provider> element to specify explicitly which persistence provider to use to access the persistence entity.
  2. Build the specific version of the third-party persistence provider into the application. To manage the persistence correctly, verify that the EntityManagerFactory and the EntityManager are calling the correct provider. Many providers write startup and version information to standard output, which gets included in the SystemOut.log of the application server. This information can be helpful to determine if a third-party provider is being used by your application.
  3. To use a third-party JPA provider that was not bundled with the application server, configure the class loader order for your application to load the classes with the application class loader first. This is also required if the third-party JPA provider is defined in a shared library that is assigned to a user-defined classloader on the server. If the class loader is not configured properly, the third-party JPA provider that is included with the application server will be loaded and used by your application.
  4. Depending on how you packaged the provider, perform one of the following steps:
     – If you packaged the provider in an EAR file, specify the third-party persistence provider binaries in the Manifest.mf classpath in those application modules that needs JPA access.
     – If the provider was bundled in a WAR file, include the necessary provider binaries in the WEB-INF/lib directory of the web application.
  5. Install your application normally.

  This configuration localizes the availability of the provider and limiting it to the application alone. While sometimes necessary, embedding persistence providers will increase the application's memory footprint accordingly.
- **Using shared libraries to implement a third-party persistence provider** Persistence providers accessed by applications in a global environment can be installed as a shared library for the application server. Depending on your requirements, you can share the library to the server and application. Use the following steps to implement a third-party persistence provider using shared libraries:
  1. Modify the <provider> element to specify explicitly which persistence provider to use to access the persistence entity.
  2. Define the persistence provider in a shared library. See the topic on creating shared libraries for more information.
  3. Associate the shared library with the application class loader, or associate the shared library with the server class loader if the library is accessed by many applications.
  4. Configure the class loader order for your application to load the classes with the application class loader first. This is required if the third-party JPA provider is included in the application or if it is

defined in a shared library. If the class loader is not configured properly, the JPA provider that is included with the application server will be used by your application instead of the third-party JPA provider.

When the application starts, the specified persistence provider will be resolved. From this point on, all persistence requests will be handled by this provider.

- If the shared library persistence provider is overriding an existing persistence provider that is the same type and exists in your application's classloader hierarchy (example: attempting to override the version of OpenJPA that is shipped with WebSphere), using an isolated classloader for the shared library is recommended. See the topic on creating shared libraries for more information.

Related tasks

Creating shared libraries
Shared libraries are files used by multiple applications. Create a shared library to reduce the number of duplicate library files on your system.

Configuring class loaders of a server
You can configure the application class loaders for an application server. Class loaders enable applications that are deployed on the application server to access repositories of available classes and resources.

Task overview: Data Studio pureQuery

Data Studio pureQuery provides Java Persistence API (JPA) users an alternative way to access a DB2 database. PureQuery supports static Structured Query Language (SQL).

About this task

Note: JPA in the Java EE and Java SE environments provides optional support for the DB2 pureQuery runtime environment. PureQuery is a high performance Java data access platform that helps manage applications that access data. PureQuery provides an alternate set of APIs that can be used instead of Java Database Connectivity (JDBC) to access the DB2 database.

To use this feature on the application server, Data Studio pureQuery runtime version 1.2 must be installed. If you plan on performing the DB2 bind command from the administrative console or with the wsadmin tool, you must have pureQuery v1.2 or later. Refer to the Data Studio information center topic on installing pureQuery Runtime for more information.

The application server offers support for static SQL for Enterprise JavaBeans (EJB) 2.x entity beans with the ejbdeploy SQLj option. With JPA, this feature is offered through pureQuery.

DB2 pureQuery offers a number of benefits over JDBC and SQLJ. Static SQL offers greater security and control over access to data because applications are only granted authority to execute known SQL. Static SQL offers better resource utilization on the DB2 server because it avoids runtime parsing and optimizing of the SQL statements.

PureQuery makes use of DB2 packages. These packages consist of information for one or more SQL statements and are stored in the DB2 catalog. To create the packages, the user must first run the wsdb2gen command on a JPA application. The wsdb2gen command creates a *persistence_unit_name*.pdqxml file. This file contains pre-generated SQL statements for Create, Update, Delete, and Retrieve, NamedQueries and NamedNativeQueries of JPA entities. The *persistence_unit_name*.pdqxml file must be bound against database. Associated DB2 packages are generated and the SQL statement is executed statically at runtime. This *persistence_unit_name*.pdqxml file must be included into the application Java archive (JAR) file.

When doing the bind process and when you define your JDBC provider, the following four JAR files must be in the class path:

- db2jcc_license_cisuz.jar

- db2jcc_license_cu.jar
- pdq.jar
- pdqmgmt.jar

WebSphere Application Server support for pureQuery

Note:

- There is no support for the `QueryTimeout` property specified either through the FetchPlan API or through the property plug-in string for `wsjpa.ConnectionFactoryProperties`. The `QueryTimeout` value is ignored if specified.
- OpenJPA large result processing uses JDBC APIs for scrollable cursors.

Note:

- JPA sets the pureQuery property, `pdq.executionMode`, to the value `STATIC`.
- In addition to the JDBC driver JAR file, the JDBC provider configuration must include the JAR file for the pureQuery runtime environment.
- OpenJPA provides support to allow application programs to programmatically access and alter the FetchPlan at run time. Altering the fetch plan might result in SQL that has not been generated by the wsdb2gen command at application build time. If this occurs, the SQL is executed dynamically rather than using static SQL from the DB2 package.
- If the user makes changes to the application queries, entity mapping or persistence properties, then run the wsdb2gen command and bind again. This process generates and binds the updated DB2 packages.
- Input parameter values in JPA queries (with both EJB SQL queries and native SQL queries) cannot be `NULL` values except in the case of update statements `SET` expression values. To search for `NULL` values in a WHERE clause of SELECT, UPDATE or DELETE, then enter the `is null` predicate instead.

- Learn how to **"Configuring to use pureQuery in a Java EE environment"**.
- Learn how to **"Configuring to use pureQuery in Java SE environment" on page 1377**.

Related tasks

"Configuring Persistence Provider support in the application server" on page 1368
Persistence providers are implementations of the Java Persistence API (JPA) specification and can be deployed in the Java EE compliant application server that supports JPA persistence.

wsdb2gen command
The command allows users to utilize the pureQuery feature in Java Persistence API (JPA) applications.

wsenhancer command
The entity enhancer tool for Java Persistence API (JPA) applications in the application server inserts bytecode into an entity class file that allows the JPA provider to manage the state of an entity.

Developing and packaging JPA applications for a Java EE environment
Containers in the application server can provide many of the necessary functions for the Java Persistence API (JPA) in a Java Enterprise Edition (Java EE) environment. The application server also provides JPA tools to assist you with developing applications in a Java EE environment.

Developing and packaging JPA applications for a Java SE environment
Prepare and package persistence applications to test outside of the application server container in a Java SE environment.

Configuring to use pureQuery in a Java EE environment

Use this task to configure the application data source Java Database Connectivity (JDBC) provider to use pureQuery to access DB2.

Before you begin

If you need to use multiple DB2 package collections, see the information center topic configure pureQuery to use multiple package collections before continuing with this task.

About this task

PureQuery makes use of DB2 packages. These packages consist of information for one or more SQL statements and are stored in the DB2 catalog. You must first run the wsdb2gen command on a JPA application to create the packages. The wsdb2gen command creates an XML file containing SQL statement information. This XML file must be included into the application Java archive (JAR) file. The DB2 bind command uses this file as input to create the DB2 package .

Note:

- JPA sets the pureQuery property `pdq.executionMode` to the value `STATIC`.
- The JDBC provider configuration must include the JAR file for the pureQuery runtime environment. This JAR file is in addition to the JDBC driver JAR file. See the information center topic on installing pureQuery runtime for more information.
- If this is an XA data source, define a new custom property on the data source where *property_name* = `downgradeHoldCursorsUnderXa` and boolean value = `true`.

1. Update the application data source JDBC provider configuration to include the pureQuery runtime JAR file. Either define an new JDBC provider or modify an existing provider to include the following JAR files. See information center topics on JDBC provider settings and installing pureQuery Runtime for more information.
   - `pdq.jar`
   - `pdqmgmt.jar`
2. Using the DB2 bind command, bind the XML file to the database. This creates the DB2 packages. There are three ways to do this:
   - Use the wsadmin command. Refer to the information center topic on application management command group for the AdminTask object for more information.
   - Using the administrative console. Refer to the information center topic on SQLj profiles and pureQuery bind files settings for more information.
   - Using the DB2 bind command provided by Data Studio. Refer to the information center topic on the pureQuery Bind utility for more information.

What to do next

If you want to reconfigure the data source for JDBC, just remove the `pdq.jar` and `pdqmgmt.jar` from the class path.

Related tasks

"Task overview: Data Studio pureQuery" on page 1373
Data Studio pureQuery provides Java Persistence API (JPA) users an alternative way to access a DB2 database. PureQuery supports static Structured Query Language (SQL).

"Configuring a JDBC provider using the administrative console" on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

Related reference

wsenhancer command
The entity enhancer tool for Java Persistence API (JPA) applications in the application server inserts bytecode into an entity class file that allows the JPA provider to manage the state of an entity.

wsdb2gen command
The command allows users to utilize the pureQuery feature in Java Persistence API (JPA) applications.

"JDBC provider settings" on page 1455
Use this page to modify the settings for a JDBC provider.

SQLj profiles and pureQuery bind files settings
Use this panel to specify customization and binding settings for the Structured Query Language in Java (SQLJ) profiles and pureQuery bind files for DB2 that are included in this application. You can view SQLJ profiles for other database types, but you cannot change the these profiles. PureQuerybind files are only valid for DB2. Use SQLJ or pureQuery to develop data access applications that connect to DB2 databases. SQLJ is a set of programming extensions that enable a programmer to use the Java programming language to embed statements that provide SQL (Structured Query Language) database requests. PureQuery provides an alternate set of APIs that can be used instead of JDBC to access the DB2 database.

Application management command group for the AdminTask object
You can use the Jython or Jacl scripting languages to manage applications with the wsadmin tool. Use the commands and parameters in the AppManagementCommands group can be used to display and process SQL-Java (SQLJ) profiles or pureQuery bind files.

Configuring pureQuery to use multiple package collections:

Set up a pureQuery Java Persistence API (JPA) application to use multiple DB2 package collections.

About this task

It is possible for multiple copies of a database schema to exist. This situation might happen in a partitioned database schema where there is one database for east coast employee data and another database for west coast employee data. In this case, the two databases have the same schema. There might be two databases with two database catalogs. Or there might be only one database, in which case, the high-level qualifier of the table names (the schema name) must be different. Since the schemas are the same, there can be a single set of JPA entities that are used to access both sets of data. There are several ways to configure JPA to handle these situations.

Note: When there are multiple persistence units, either with separate databases or a single database, you must run the wsdb2gen command once for each persistence unit.

The following three scenarios exist that require the use of multiple DB2 package collections. If you need more information on configuring pureQuery, read about configuring an application to use pureQuery.

1. When there are two persistence units with different data source names, using static SQL, two sets of DB2 packages exist: one DB2 package in each database. Since two persistence units exist, two *persistence_unit_name*.pdqxml files for the JPA runtime environment exist.
2. If the tables are in a single database, then two persistence units can also be used. In this case, the data source is the same in both persistence units. However, the schema name property, wsjpa.jdbc.Schema must be different. There are two sets of DB2 packages. Each DB2 package must

have a different package name or a different package collection name. Both the wsdb2gen and the DB2 bind command have options to specify the package collection and package names.

3. You can create a single persistence unit, which will eliminate the need to maintain two persistence unit configurations and run the wsdb2gen command multiple times. This configuration requires a common package name. Thus the package collection names must be different. Use the createEntityManager(Map map) method, where the map contains the values for the `wsjpa.jdbc.Schema` and `wsjpa.jdbc.CollectionId` properties to specify the package collection name and schema name.

Related tasks

"Configuring to use pureQuery in a Java EE environment" on page 1374
Use this task to configure the application data source Java Database Connectivity (JDBC) provider to use pureQuery to access DB2.

Related reference

wsdb2gen command
The command allows users to utilize the pureQuery feature in Java Persistence API (JPA) applications.

Configuring to use pureQuery in Java SE environment

Use this task to configure the application data source Java Database Connectivity (JDBC) provider to use pureQuery to access DB2.

Before you begin

If you need to use multiple DB2 package collections, see the information center topic configure pureQuery to use multiple package collections before continuing with this task.

About this task

PureQuery makes use of DB2 packages. These packages consist of information for one or more Structured Query Language (SQL) statements and are stored in the DB2 catalog. You must first run the wsdb2gen command on a Java Persistence API (JPA) application to create the packages. The wsdb2gen command creates an XML file containing SQL statement information. This XML file must be included into the application Java archive (JAR) file. The DB2 bind command uses this file as input to create the DB2 package .

Note:

- JPA sets the pureQuery property `pdq.executionMode` to the value `STATIC`.
- The class path must include the install location for the pureQuery Runtime. See the information center topic on installing pureQuery Runtime for more information.
- The JPA provider implementation must be JPA for the application server (`com.ibm.websphere.persistence.PersistenceProviderImpl`). The OpenJPA persistence provider does not provide support for pureQuery.
- The wsdb2gen command requires the URL of a database. The wsdb2gen command forces a synchronize mapping function that creates or alters the required tables. For DB2 zOS, V8 unique indexes and LOB tables must be manually created prior to executing the wsdb2gen command.

PureQuery properties are specified in a `pdq.properties` file in the META-INF directory of the application jar. The `pdq.ExecutionMode` property is defaulted to STATIC for JPA applications. PDQ properties, if specified, pass on to the pureQuery runtime. See Data Studio pureQuery runtime documentation for list of properties and valid values.

- **wsjpa.jdbc.CollectionId** : String value specifying the collection id to use. This overrides any collection id used during wsdb2gen.

1. Update the application data source JDBC provider configuration to include the pureQuery Runtime JAR files. Include the `pdq.jar` and `pdqmgmt.jar` files on the class path in addition to the JDBC driver jar files. Either define an new JDBC provider or modify an existing provider to include the JAR files. The

class path must include the install location for the pureQuery Runtime. See the information center topics on JDBC provider settings and installing pureQuery Runtime for more information.

2. Using the DB2 bind command provided by Data Studio, bind the XML file to the database. This creates the DB2 packages. Refer to the information center topic on the pureQuery Bind utility for more information.

What to do next

If you want to reconfigure the data source for JDBC, remove the `pdq.jar` and `pdqmgmt.jar` from the class path.

Related tasks

"Task overview: Data Studio pureQuery" on page 1373
Data Studio pureQuery provides Java Persistence API (JPA) users an alternative way to access a DB2 database. PureQuery supports static Structured Query Language (SQL).

"Configuring a JDBC provider using the administrative console" on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

Related reference

wsenhancer command
The entity enhancer tool for Java Persistence API (JPA) applications in the application server inserts bytecode into an entity class file that allows the JPA provider to manage the state of an entity.

wsdb2gen command
The command allows users to utilize the pureQuery feature in Java Persistence API (JPA) applications.

"JDBC provider settings" on page 1455
Use this page to modify the settings for a JDBC provider.

SQLj profiles and pureQuery bind files settings
Use this panel to specify customization and binding settings for the Structured Query Language in Java (SQLJ) profiles and pureQuery bind files for DB2 that are included in this application. You can view SQLJ profiles for other database types, but you cannot change the these profiles. PureQuerybind files are only valid for DB2. Use SQLJ or pureQuery to develop data access applications that connect to DB2 databases. SQLJ is a set of programming extensions that enable a programmer to use the Java programming language to embed statements that provide SQL (Structured Query Language) database requests. PureQuery provides an alternate set of APIs that can be used instead of JDBC to access the DB2 database.

Application management command group for the AdminTask object
You can use the Jython or Jacl scripting languages to manage applications with the wsadmin tool. Use the commands and parameters in the AppManagementCommands group can be used to display and process SQL-Java (SQLJ) profiles or pureQuery bind files.

Configuring pureQuery to use multiple package collections:

Set up a pureQuery Java Persistence API (JPA) application to use multiple DB2 package collections.

About this task

It is possible for multiple copies of a database schema to exist. This situation might happen in a partitioned database schema where there is one database for east coast employee data and another database for west coast employee data. In this case, the two databases have the same schema. There might be two databases with two database catalogs. Or there might be only one database, in which case, the high-level qualifier of the table names (the schema name) must be different. Since the schemas are the same, there can be a single set of JPA entities that are used to access both sets of data. There are several ways to configure JPA to handle these situations.

Note: When there are multiple persistence units, either with separate databases or a single database, you must run the wsdb2gen command once for each persistence unit.

The following three scenarios exist that require the use of multiple DB2 package collections. If you need more information on configuring pureQuery, read about configuring an application to use pureQuery.

1. When there are two persistence units with different data source names, using static SQL, two sets of DB2 packages exist: one DB2 package in each database. Since two persistence units exist, two *persistence_unit_name*.pdqxml files for the JPA runtime environment exist.
2. If the tables are in a single database, then two persistence units can also be used. In this case, the data source is the same in both persistence units. However, the schema name property, wsjpa.jdbc.Schema must be different. There are two sets of DB2 packages. Each DB2 package must have a different package name or a different package collection name. Both the wsdb2gen and the DB2 bind command have options to specify the package collection and package names.
3. You can create a single persistence unit, which will eliminate the need to maintain two persistence unit configurations and run the wsdb2gen command multiple times. This configuration requires a common package name. Thus the package collection names must be different. Use the createEntityManager(Map map) method, where the map contains the values for the wsjpa.jdbc.Schema and wsjpa.jdbc.CollectionId properties to specify the package collection name and schema name.

Related tasks

“Configuring to use pureQuery in a Java EE environment” on page 1374
Use this task to configure the application data source Java Database Connectivity (JDBC) provider to use pureQuery to access DB2.

Related reference

wsdb2gen command
The command allows users to utilize the pureQuery feature in Java Persistence API (JPA) applications.

Associating persistence units and data sources

Java Persistence API (JPA) applications allow you to specify the underlying data source used by the persistence provider to access the database.

About this task

The application server provides three methods for defining the data sources in the persistence.xml file.

- Explicitly specify the Java Naming and Directory Interface (JNDI) name in the persistence.xml file, and the application will directly reference the data source. Switching to another data source requires an update to the persistence.xml file.

  JPA has two transactional patterns for accessing a data source:

  – The Java Transaction API (JTA) resource pattern depends on global transactions. The JTA resource pattern is typically used within the scope of an Enterprise JavaBeans (EJB) session facade. This allows the session bean to control transaction and security contexts while JPA handles the persistence mappings. In this case, the application does not use the EntityTransaction interface but relies on the EntityManager enlisted with the global transaction when it is accessed.

  – The non-JTA resource pattern is generally used when dealing with a single resource in the absence of global transactions. The non-JTA resource pattern is typically used within the scope of a web application or an application client. The application controls the transaction with the data source with the EntityTransaction interface.

    Within the application server, the use of the <non-jta-data-source> element requires a special configuration for a non-transactional data source. Data sources that are configured for the application server will not function as a <non-jta-data-source>, because all data sources that are configured by the application server are automatically enlisted with the current transactional context. To prevent this automatic enlistment, add an additional data source custom property nonTransactionalDataSource=true:

1. Select **Resources** → **JDBC** → **Data sources**
2. Select the name of the data source that you want to configure.
3. Select **WebSphere Application Server data source properties** from the **Additional Properties** heading.
4. Select **Non-transactional data source**.
5. Click **OK**.

Note: The JPA specification assumes that connections will be obtained with an isolation level that does not hold long term locks in the database, such as `READ_COMMITTED`. This may not match the WebSphere Application Server default isolation level, which is `REPEATABLE_READ` for most databases. You can check the level which is used for your database by referring to the topic Requirements for setting isolation level.

If the default for your database is not `READ_COMMITTED`, you may change the default by adding an additional data source custom property `webSphereDefaultIsolationLevel`. The following table shows the valid values.

Table 9. Isolation level values

Value	Isolation Level
1	READ_UNCOMMITTED
2	READ_COMMITTED (JPA default)
4	REPEATABLE_READ (WebSphere Application Server default)
8	SERIALIZABLE

If the isolation level is set to a value that will hold long-term read locks, you need to configure the JPA provider to use Pessimistic Locking instead of the default Optimistic Locking. For the JPA provider included with WebSphere Application Server, you may do this by adding the following properties to `persistence.xml` file:

```
<property name="openjpa.Optimistic"="false"/>
<property name="openjpa.LockManager"=pessimistic"/>
```

The JPA specification mandates that the data sources that are defined in the `<jta-data-source>` and `<non-jta-data-source>`elements of a persistence unit register in the JNDI name space. For example, the persistence.xml file should contain an entry similar to the following:

```
<jta-data-source>jdbc/FooBarDataSourceJNDI</jta-data-source>
```

- The JPA solution for the application server extends the implementation to allow you to reference data sources in component name space. In the EJB or Web module deployment descriptor file, this is the <resource-ref> element. You can prefix the data source with java:comp/env/ so the application indirectly references the data source by using the local JNDI name. In this association, the application does not require updates, you change the <resource-ref> to use another data source. See the following example:

  ```
  <jta-data-source>java:comp/env/jdbc/FooBarDataSourceJNDI</jta-data-source>
  ```

  Note: JPA specifications do not require implementations to include data sources referenced through local JNDI names, so this method of reference may not be portable to other implementations of JPA.

- You can declare openjpa.Connection* properties in the persistence unit. If you are not using JPA inside the application server, you need to use this method:

  Example

  ```
  <property name="openjpa.ConnectionDriverName" value="org.apache.derby.jdbc.EmbeddedDriver" />
  <property name="openjpa.ConnectionURL" value="jdbc:derby:target/database/jpa-test-database;create=true"/>
  ```

What to do next

For information about configuring data sources, see the topic on creating and configuring a data source.

For information about data sources and JPA, see the section on persistence in the Apache OpenJPA User's Guide.

Configuring OpenJPA caching to improve performance

The OpenJPA implementation allows users the option of storing frequently used data in the memory to improve performance. OpenJPA provides concurrent data and concurrent query caches that allow applications to save persistent object data and query results in memory to share among threads and for use in future queries.

About this task

OpenJPA data cache functionality

The OpenJPA data cache is a cache of persistent object data that operates at the EntityManagerFactory level. This optional-use cache is designed to increase performance while remaining in full compliance with the Java Persistence API (JPA) standard. This means that enabling the caching option can increase the performance of your application, with no changes to your code. The OpenJPA data cache is designed to provide significant performance increases over cacheless operations and ensures that behavior will be identical in both cache-enabled and cacheless operations.

When enabled, the cache is examined before accessing the datastore. The cache stores data when objects are committed and when persistent objects are loaded from the datastore. If operating in a single Java virtual machine (JVM) environment, the JVM maintains and shares a data cache across all EntityManager instances obtained from a particular EntityManagerFactory. The OpenJPA data cache cannot do this in a distributed environment because caches in different JVMs, created from different EntityManagerFactory objects will not be synchronized. Using the OpenJPA cache in a multi-JVM environment requires either using the event notification framework or through custom integrations with a third-party distributed cache utilities such as IBM ObjectGrid.

Enabling and configuring the OpenJPA data cache

You can enable the OpenJPA data cache for a single or a multiple JVM environment, set its default element size, including soft references, and specify timeout values.

To set up and configure the OpenJPA data cache, do the following:

1. To enable the cache for a single JVM, set the openjpa.DataCache property to true, and set the openjpa.RemoteCommitProvider property to sjvm:

   ```
   <property name="openjpa.DataCache" value="true"/>
   <property name="openjpa.RemoteCommitProvider" value="sjvm"/>
   ```

   To enable the data cache in a distributed environment, the openjpa.RemoteCommitProvider must be configured specifically for the environment, or a third-party cache management utility can be used.
2. The maximum cache size can be adjusted by setting the CacheSize property:

   ```
   <property name="openjpa.DataCache" value=true(CacheSize=5000...
   ```

   By default, the OpenJPA data cache holds 1000 elements. Objects that are pinned into the cache are not counted when determining if the cache size exceeds its maximum size. If the cache overflows, it evicts random elements. You can preserve evicted elements longer with the SoftReferenceSize property. By default, soft references are unlimited. If you need to, you can limit the number of soft references or set to 0 to disable soft references completely:

   ```
   <property name="openjpa.DataCache" value="true(CacheSize=5000 SoftReferenceSize=0 ...
   ```
3. You can specify that a cache should be cleared at certain times. The EvictionSchedule property of the OpenJPA cache implementation accepts a cron style eviction schedule. The cron format specifies the

minute, hour of day, day of month, day of month, and day of the week beginning with 1 for Sunday; the * symbol (asterisk), indicates match all. To schedule a cache to evict at 45 minutes past 3 PM on Sunday every month you would add this property:

```
<property name="openjpa.DataCache" value="true(CacheSize=5000 SoftReferenceSize=0 EvictionSchedule='15,45 * * 1'")/>
```

Note: You also can specify a cache timeout value for a single class by setting the timeout metadata extension to the amount of time in milliseconds that the data of the class is valid. Refer to the org.apache.openjpa.persistence.DataCache Javadoc for more information.

After configuring your data cache, you can use it after you restart your application.

Refresh with active DataCache

Refreshing an entity may lead to different behavior with or without a DataCache when a separate process or part of the same application are updated or even deleted the corresponding record in the database. By default, entities are refreshed from the database even when DataCache is active. Therefore, with the default configuration the refresh behaves identically with or without a DataCache. However, a persistence unit can be configured to refresh entities from DataCache with the property `openjpa.RefreshFromDataCache` for improved performance. Under this configuration, any out-of-band changes that occur in the database record will not appear in the refreshed state of the entity.

Note: Regardless of the `openjpa.RefreshFromDataCache` setting, the DataCache is always bypassed for refresh when locks are active, such as for a pessimistic transaction, in a persistence context. An application may activate `openjpa.RefreshFromDataCache` but can still bypass the DataCache while refreshing an entity by explicitly evicting the entity from DataCache prior to refresh.

Query Caching functions

OpenJPA provides a concurrent query cache that allows applications to save persistent object data and query results in memory to share among threads and for use in future queries. The query cache stores the object ids returned by query executions. When you run a query, OpenJPA assembles a key based on the query properties and the parameters used at execution time, and checks for a cached query result. If one is found, the object ids in the cached result are looked up, and the resultant persistence-capable objects are returned. Otherwise, the query is executed against the database, and the object ids loaded by the query are put into the cache.

Configuring or disabling the query cache

You can configure the query cache settings in a similar way to the data cache. The interface provided to the query cache is the org.apache.openjpa.persistence.QueryResultCache class. You can access this class through the OpenJPAEntityManagerFactory.

The default query cache implementation caches 100 query executions in a least-recently-used cache. This can be changed by setting the cache size in the CacheSize plugin property. Like the data cache, the query cache also has a backing soft reference map that can be changed using the SoftReferenceSize property. To keep queries in the cache at all times, you can pin them to a cache. To change the query cache properties do the following:

1. Modify the CacheSize property of the openjpa.QueryCache:

   ```
   <property name="openjpa.QueryCache" value=("CacheSize=1000, ...
   ```

2. Change the SoftReferenceSize property to enable and control the size of this map:

   ```
   <property name="openjpa.QueryCache" value=("CacheSize=1000, SoftReferenceSize=100")/>
   ```

   The SoftReferenceSize table is disabled by default. Setting the size enables it.

3. Pin or unpin queries in the cache through the QueryResultCache with this syntax:

```
public void pin(Query q);
public void unpin(Query q);
```

Modifying these properties allows you to make better use of the query cache.

Extending a cache

OpenJPA provides classes that may be extended for further functionality.

- As previously mentioned, if you want to implement a distributed cache that uses an unsupported method for communications, create an implementation of org.apache.openjpa.event.RemoteCommitProvider.
- If you are adding new behavior, you should extend org.apache.openjpa.datacache.DataCacheImpl.
- To use your own storage mechanism, extend org.apache.openjpa.datacache.AbstractDataCache.
- To add query functionality, you can extend the default org.apache.openjpa.datacache.QueryCacheImpl.
- Implement your own storage mechanism for query results by extending org.apache.openjpa.datacache.AbstractQueryCache

OpenJPA query SQL cache

OpenJPA provides a cache that provides caching of SQL strings used by find operations performed on the entity manager and some queries to manage eagerly fetched relationships. When this cache is enabled, SQL queries used by these operations are generated once per entity manager factory and may be reused. This cache is enabled by default but can also be configured through the `openjpa.jdbc.QuerySQLCache` configuration property.

Configuring or disabling the SQL query cache

The query SQL cache can be configured or disabled through the `openjpa.jdbc.QuerySQLCache` property. By default, this property is set to true. When set to true the cache is enabled and uses the `org.apache.openjpa.util.CacheMap` class for its cache store. The CacheMap is a managed cache, meaning that it limits the number of cache entries and has a cache eviction scheme to manage memory usage. If the cache is set to all the `org.apache.openjpa.lib.util.ConcurrentHashMap` class is used as a cache store. The ConcurrentHashMap is not a managed cache so entries will remain in the cache for the lifetime of an entity manager factory. This caching mechanism may provide better performance at the expense of increased memory usage. A custom cache store class can also be specified provided that it implements the java.util.Map interface. To disable the cache, specify the value false. See the following examples on how to configure or disable the SQL query cache:

- To use an unmanaged cache:
  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="all"/>
  ```
- To specify a custom cache class:
  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="com.mycompany.MyCustomCache"/>
  ```
- To use an unmanaged cache:
  ```
  <property name="openjpa.jdbc.QuerySQLCache" value="false"/>
  ```

What to do next

Refer to chapter 10, Caching in the OpenJPA reference for all caching extensions.

Related tasks

Developing and packaging JPA applications for a Java SE environment
Prepare and package persistence applications to test outside of the application server container in a Java SE environment.

Troubleshooting Java Persistence API (JPA) applications

Use this information to find various known problems with JPA applications.

- Review messages that are related to transactions.

  Under certain conditions, a message similar to the following might be logged:

  ```
  Unable to execute {0} on a WebSphere managed transaction. WebSphere does not support direct manipulation of managed transactions.
  ```

  This error is probably caused by a data source that is not configured correctly as <non-jta-data-source>. See the information center topic on how to configure a data source to be used as a <non-jta-data-source>.

- Troubleshoot classes that have not been enhanced by run time.

  It is difficult to diagnose these situations. Sometimes you can trace the problem back to a lack of OpenJPA enhancement of entity classes. Examples of these situations might be detecting when entities are not ″persistence capable″. Look for a message similar to the following in the log:

  ```
  This configuration disallows runtime optimization, but the following  listed types were not enhanced at build time or at class load time  with a javaagent: "{0}".
  ```

  This message indicates that the expected runtime enhancement did not take place on the listed entity types. In most cases, this is just a build time failure and the PCEnhancer needs to be run on the listed classes. It can also indicate a more involved problem, especially if you are expecting the container classloader transform to do the entity enhancement.

- Troubleshoot issues with closed cursors.

  The following is an exception from DB2 in the log org.apache.openjpa.persistence.PersistenceException:

  ```
  [ibm][db2][jcc][10120][10898] Invalid operation: result set is closed can be a WebSphere Application Server configuration problem.
  ```

  By default, the application server configures the resultSetHoldability custom property with a value of 2 (CLOSE_CURSORS_AT_COMMIT). This property causes DB2 to close its resultSet/cursor at transaction boundaries. Despite DB2's default resultSetHoldability value of 1 (HOLD_CURSORS_OVER_COMMIT), the application server retains the default value of 2 to avoid breaking compatibilities with previous releases of the application server. You can change the default if the need arises.

  Note: If this is an XA datasource, define a new custom property on the datasource where `property_name = downgradeHoldCursorsUnderXa` and `boolean value = true`.

  Note: In Version 7.0, if a DB2 XA Data source is used, the following configuration are required to overcome the ″result set closed″ problem:

  1. Apply DB2 Version 8.1 FP17 for APAR IZ18070 or Version 9 FP4 for APAR IZ18072.
  2. Add custom property `name="downgradeHoldCursorsUnderXa"`, `value="true"` and `type="java.lang.Boolean"`
  3. Add custom property `name="resultSetHoldability"`, `value="1"`, and `type="java.lang.Integer"`

- Avoid multi-threaded EntityManager usage. If you experience an exception with the following message text, you might be accidentally allowing the usage of an EntityManager across multiple threads.

Per the JPA specification, EntityManagers are meant to be used by a single thread. You might receive an exception message that is similar to the following (in this context, brokers and EntityManagers are essentially the same thing):

```
Multiple concurrent threads attempted to access a single broker. By default brokers are not
thread safe; if you require and/or intend a broker to be accessed by more than one thread,
set the openjpa.Multithreaded property to true to override the default behavior.
```

There are some ways to address this issue:

– As documented in this exception text, an easy workaround to the situation is to tell OpenJPA that you are requiring multi-threaded access to the EntityManagers through the openjpa.Multithreaded property. This can introduce unnecessary overhead if multi-threaded access occurs that is not intentional.

–

Note: Utilize the get-use-close pattern when using application-managed (extended) persistence contexts. Remember to close the EntityManager before leaving the method that used the EntityManager. Leaving the EntityManager open can accidentally allow other threads to use the same EntityManager instance at the same time.

– Change the application to use container-managed persistence contexts by injecting the EntityManager instance through the @PersistenceContext annotation, if the programming model for the application is amenable to this change. Essentially, this enforces the get-use-close pattern by allowing the container to do the management.

• Be aware of versioning conditions if you are using optimistic locking. In the JPA architecture, the persistence provider uses the version property to perform optimistic locking and concurrency semantics for a persisting entity. For example:

```
@Entity
public class VersionedTimestampEntity
{     @Id      private int id;
      @Version private java.sql.Timestamp version;
       ....
}
```

The provider updates the versioned property to a new value when an entity is written to the database. During an entity merge operation, the persistence provider examines this versioned property to determine if the entity that is being merged is a stale copy of the entity. If the operation failed due to the stale versioning condition, an OptimisticLockException will be thrown. The versioned property can be one of these types:

– int
– Integer
– short
– Short
– long
– Long
– Timestamp.

Note: If you use the Timestamp type for the versioned property, the application needs to be aware of behavior that can be exhibited due from the implementation precision that is used for creating the Timestamp object. The typical implementation uses System.currentTimeMills(), and the time precision of this method is platform specific. For example, the precision is approximately 15ms for 32-bit Windows and 2ms for z/OS.

If an entity is persisted, and the entity is then fetched and updated in a separate persistence context that is within the precision window of the platform, the persistence provider will not be able to detect the optimistic locking and concurrent condition. As a result, an OptimisticLockException may not be thrown and data integrity could be compromised.

- Troubleshoot database constraint violations when working with entities.

  The JPA providers included with WebSphere Application Server use a constraint update manager to determine the order of SQL requests to the database based on each entity's configuration. This update manager may rearrange the order of SQL requests to the database. This alleviates an application from knowing the order of entity manager operations required to perform its business logic and optimizes database performance using underlying batching support.

  Since the JPA provider does not assume that there are implied database constraints for relationships between entities, if there are constraints in the database (for example, a foreign key constraint) the JPA provider may not issue SQL statements in the desired order. Under these conditions, the following or similar exception may be observed:

  ```
  com.ibm.db2.jcc.b.SqlException: DB2 SQL error: SQLCODE: -530, SQLSTATE: 23503, SQLERRMC: xxxxxx
  ```

  There are some ways to address this issue:

  – The JPA provider can be configured to read constraint information from the database and apply to the update manager at runtime by adding the following configuration property to the persistence unit.

    ```
    <property name="openjpa.jdbc.SchemaFactory" value="native(ForeignKeys=true)" />
    ```

  – The application can enhanced entity to use @ForeignKey annotation to indicate to the JPA provider which relationships have foreign key constraints.

    ```
    public class Person {
         @ManyToOne
         @ForeignKey
         private Address address;
         ....
     }
    ```

  – The application can take on the responsibility of ordering the SQL statements by adding the following configuration property to the persistence unit.

    ```
    <property name="openjpa.jdbc.UpdateManager" value="operation-order" />
    ```

    With this configuration option present, the JPA provider will execute the SQL statements in the same order as the entity operations were requested. The application must be aware of any interdependencies between entities.

  – Remove the constraints from the database.

- Troubleshoot using the OpenJPA MappingTool ANT task.

  The MappingTool ANT task provided by OpenJPA uses a temporary classloader to load the JDBC driver classes. This temporary classloader may have trouble loading some JDBC drivers such as DB2.

  When you run the MappingTool ANT task, you could see something similar to the following error:

  ```
  [mappingtool] java.lang.UnsatisfiedLinkError: com/ibm/jvm/Trace.initTrace([Ljava/lang/
  String;[Ljava/lang/String;)V [mappingtool] at
  com.ibm.jvm.Trace.initializeTrace(Trace.java:94) [mappingtool] at
  com.ibm.jvm.Trace.<clinit>(Trace.java:59) [mappingtool] at
  java.lang.J9VMInternals.initializeImpl(Native Method) [mappingtool] at
  java.lang.J9VMInternals.initialize(J9VMInternals.java:200) [mappingtool] at
  java.lang.Class.forNameImpl(Native Method) [mappingtool] at
  java.lang.Class.forName(Class.java:136) [mappingtool] at com.ibm.db2.jcc.a.o.n(o.java:577)
  [mappingtool] at com.ibm.db2.jcc.a.o.<clinit>(o.java:329)
  ```

  In order to use the mapping tool, you can disable the temporary classloader by adding the tmpClassLoader=false argument to the ANT task. Two example ANT scripts follow :

  This example will exhibit the problem:

  ```
  <taskdef name="mapping" classname="org.apache.openjpa.jdbc.ant.MappingToolTask" classpathref="jpa.cp"/>
      . . .
   <target name="map.broken">
       <mapping>
           <!-- this will exhibit the problem -->
           . . .
       </mapping>
   </target>
  ```

This example will prevent the problem:

```
<taskdef name="mapping" classname="org.apache.openjpa.jdbc.ant.MappingToolTask" classpathref="jpa.cp"/>
    . . .
 <target name="map.fixed">
    <mapping tmpClassLoader="false">
        <!-- this will work -->
        . . .
    </mapping>
</target>
```

- Impact of DataCache on inconsistent domain models

  When an application persists an inconsistent domain model and later retrieves the entities in a separate persistence context, the retrieved entities will differ depending on whether DataCache is active or not.

  For example, consider bi-directional, one-to-one relationship between two entities mapped by a single foreign key column in the database. It is possible for an application to set the relation fields in the entities inconsistently. When such inconsistent values are mapped to the database, the database records become consistent only because a single foreign key column expresses the bi-directional relationships DataCache is active, however, then DataCache captures the inconsistent in-memory entity states. Therefore, when an application persists a pair of entities related in a bi-directional relation but their relation fields are set to inconsistent values and later retrieves the entities in a different persistence context, the bi-directional relationship either remains inconsistent or becomes consistent depending on whether DataCache is used or not.

  When multiple fields are mapped to the same column but set to different values is another example where an inconsistent entity state is persisted but retrieved as consistent in a separate persistence context. In this case, the introduction of DataCache will also cause an entity realized in a separate persistence context to retain different and inconsistent values while without a DataCache, an entity will hold the same value for both the fields.

  Note: The best practice is to avoid populating the application domain model inconsistently. It is also possible to configure `openjpa.InverseManager` property to detect certain inconsistencies such as bi-directional relationship.

Related tasks

Task overview: Storing and retrieving persistent data with the Java Persistence API (JPA)
The Java Persistence API (JPA) for the application server defines the management of persistence and object/relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments.

Logging with the Java Persistence API (JPA) and the application server

Logging allows you to view, trace, and troubleshoot the runtime behavior of an application. JPA provides a flexible logging system that is integrated with the application server to assist you in troubleshooting problems.

About this task

Logging channels

Note: Logging can have a negative impact on performance. Limit or disable logging when you run any performance tests.

In additional to the channels used by OpenJPA, a trace group named openjpa enables channels that are prefixed with “openjpa.” Specifying “openjpa” for a trace group will override any other trace group specification that is specific to a channel. **Example**

```
openjpa.Runtime=debug:openjpa.jdbc.SQL=all

    openjpa=all
```

Note: The openjpa.Log property will be ignored if it is defined in a container-managed persistence unit that uses the persistence providers that are provided with the application server. In this case, you must use the standard trace specification for the application server.

OpenJPA and JPA for WebSphere Application Server implement logging channels to which message data, trace data, and debugging data can be recorded to a configurable repository. The JPA component creates the logging channel at run time and assigns a channel name for identification. The component writes information to the configured repository through the channel. OpenJPA and JPA for WebSphere Application Server create the following channels:

- **openjpa.Tools** - Command line and Ant server tools
- **openjpa.MetaData** - Meta data information
- **openjpa.Enhance** - Enhancement and runtime class generation
- **openjpa.Runtime** - Messages generated during runtime
- **openjpa.Query** - Query information
- **openjpa.DataCache** - L2 data cache plugin information
- **openjpa.jdbc.JDBC** - JDBC connection information
- **openjpa.jdbc.SQL**- Detailed SQL execution statements and information
- **wsjpa.pdq**- Trace of all interactions between store manager and PDQ runtime
- **wsajpa.Sqlgen**- Diagnostic trace for wsdb2gen program

Logging levels

Each of the logging channels use logging levels to control which messages are recorded. The following logging levels are supported by the JPA architecture:

- **TRACE** - the most detailed option
- **INFO** - information related to the specific channel
- **WARN** - warning messages
- **ERROR** - error condition messages
- **FATAL** - fatal condition messages

By using a particular logging channel together with logging levels, you can control the types of logging messages and the amount of logging messages that are recorded.

Note: These logging functions only apply to OpenJPA and JPA for the application server. Logging functions that are provided in implementations of a third-party persistence provider are not covered. However, if the logging output from a third-party persistence provider is directed to the Java System.out or System.err output streams, the messages are handled by the environment accordingly at run time.

Logging in the application server

The default JPA persistence provider that is supported by the application server records messages and tracing data that are automatically integrated into the RAS component. Alternatively, OpenJPA implements a custom logger to route messages from OpenJPA channels to the channels of the application server.

The channel names that are supported by OpenJPA are used as the trace group names in the trace level for the application server. The mappings of OpenJPA logging levels to trace levels in the application server are:

OpenJPA logging level	Trace level for the application server
TRACE	debug
INFO	info

OpenJPA logging level	Trace level for the application server
WARN	warning
ERROR	error
FATAL	fatal

Logging in a client container and standalone Java application

OpenJPA logging uses the basic logging framework and output format:

```
millis [persistence-unit name]level[thread identifier] channel - message
```

.

Note: When using pureQuery, the PDQ store manager also uses JDBC in some situations, such as for large result set processing. When tracing all calls to the database, you will need to trace both JDBC and PDQ.

Example:

```
property name="wsjpa.Log" value="SQL=TRACE"/
```

This will trace the SQL and input parameter values.

```
property name="wsjpa.Log" value="pdq=TRACE, JDBC=TRACE"/
```

This will perform a detailed trace of calls to the pureQuery runtime as well as any calls to JDBC. If you are using pureQuery and need to trace calls to the database, you will need to perform both traces.

The default logging system accepts the following parameters:

- **File**: The name of the file to which the application server will log information. You can also use standard output `stdout` or standard error `stderr` to send messages. By default, JPA sends log messages to standard error.
- **DefaultLevel**: The default logging level for channels that are not configured. The values can be `TRACE`, `INFO`, `WARN`, and `ERROR`. The default setting is `INFO`.
- **DiagnosticContext**: A string that will be placed at the beginning of all log messages. If a DiagnosticContext is not supplied and an openjpa.Id property is available, the openjpa.Id value is used.
- **<channel>**: The channel that is being logged. This parameter can be used to configure the logging level of the channel.

1. Open the persistence.xml file for the application that you want to modify.
2. Add a property name tag to the XML schema named openjpa.log. For example:
   ```
   <property name="openjpa.log" .../>
   ```
3. Add additional parameters. For example:
   ```
   <property name="openjpa.log" value="DefaultLevel=WARN .../>
   ```

   Note: To reduce overhead by disabling logging, set the openjpa.log property to NONE and proceed to Step 5.
4. Designate the logging channels and the logging level. For example:

```
<property name="openjpa.log" value="DefaultLevel=WARN, Runtime=INFO, Tool=INFO" .../>
```

5. Save changes to the file.

Results

The next time the application is started the JPA component will log all channels at the `WARN` logging level and the Runtime and Tool channels at the `INFO` level.

What to do next

OpenJPA allows users to substitute other logging methods. Consult the logging section of the Apache OpenJPA User's Guide for more information and examples.

Related tasks

Task overview: Storing and retrieving persistent data with the Java Persistence API (JPA)
The Java Persistence API (JPA) for the application server defines the management of persistence and object/relational mapping within Java Enterprise Edition (Java EE) and Java Standard Edition (Java SE) environments.

Enabling Enhanced Tracing for Java Persistence API (JPA)

In some situations the trace information generated by the JPA providers shipped with WebSphere Application Server may not be adequate to diagnose a problem. In these situations, an extended trace mechanism can be enabled to generate additional information in the trace file. Extended trace will only function with IBM-shipped persistence providers. It will not work with 3rd party providers, including versions of OpenJPA bundled within an application or configured as a shared library.

About this task

Enabling enhanced tracing in a WebSphere Application Server environment

Enhanced JPA tracing for an application running on WebSphere Application Server can be enabled with a few simple steps using wsadmin scripting or the administration console. The steps below describe how to configure enhanced tracing using the administration console. This process changes server settings so it is good practice to backup your server configuration before proceeding.

1. Enable the trace agent. A trace agent must be enabled per application server by passing an argument to the server Java virtual machine (JVM). The trace agent can be enabled using the Administration console by following these steps:
   a. In the navigation pane, select **Servers**. Select **Application Servers**.
   b. In the server list pane, select the server that needs the enhanced JPA trace. If multiple servers provide JPA functionality to your application, these steps must be followed for every server.
   c. Under the Server Infrastructure heading, select **Java and Process Management**. Select **Process Definition**.
   d. Under the Additional Properties heading, select **Java virtual machine**.
   e. Add the following argument to the Generic JVM arguments field, where <WAS install path> is the fully qualified path of your application server install directory. Make sure to use the path separator character appropriate for your operating system.

      ```
      -javaagent:<WASinstall path>/optionalLibraries/IBM/wsjpa/wsjpatrace.jar
      ```
2. Enable additional trace components and adjust trace file options. You can accomplish this with wsadmin scripting or the Administration Console. These steps describe how to adjust trace file settings and enable components using the Administration Console:
   a. In the navigation pane, select **Troubleshooting**. Click **Logs and Trace**.
   b. Select the name of the server to trace.
   c. Under the General Properties heading, select**Diagnostic Trace**.
   d. Make sure **Enable Log** is checked and optionally increase the Maximum File Size and Maximum Number of Historical Files. Depending on the number of additional trace categories and the trace levels chosen, the trace file can become extremely large.
   e. Under the Additional Properties heading, select **Change Log Detail Levels**.

      Note: The use of generic JVM arguments in the Administrative console does not currently support spaces within arguments. If spaces are specified in this field, the server may fail to start. This is more likely to occur in a Windows environment because the default install path is

C:\Program Files\IBM\WebSphere\AppServer, which contains a space in the path. To work around this problem in a Windows environment use an abbreviated path name for the <WAS install path>. For example, C:\Progra~1\IBM\WebSphere\AppServer. On Unix-type systems a symbolic link can be used to eliminate spaces in the <WAS install path>. For example, if the WebSphere Application Server installation path is /opt/WAS Install/AppServer, a symbolic link can be created in /opt from WAS Install to WASInstall, eliminating the space. Then, specify /opt/WASInstall/AppServer as the <WAS install path> in the generic JVM argument.

f. Enable various extended trace categories by specifying one or more trace categories from the table below. An example trace string is: *=info:JPA=all:openjpa.*=finer:openjpa.kernel=finest. Extended trace traces at the FINER or FINEST trace levels. The FINEST level includes more detail than FINER. When ALL is specified, extended trace traces at the FINEST level.

Category	Relevant trace levels	Description
JPA	OFF, ALL, FINER, FINEST	Adds extended trace to the JPA trace group.
openjpa.*	OFF, ALL, FINER, FINEST	Normal OpenJPA trace in addition to extended trace for all categories in OpenJPA when extended trace is enabled.
openjpa.xtrace.*	OFF, ALL, FINER, FINEST	Extended trace for all categories in OpenJPA when extended trace is enabled.
openjpa.xtrace.Jdbc	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA JDBC classes when extended trace is enabled.
openjpa.xtrace.Lib	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA library classes when extended trace is enabled.
openjpa.xtrace.Persist	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA persistence classes when extended trace is enabled.
openjpa.xtrace.Kernel	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA kernel classes when extended trace is enabled.
openjpa.xtrace.General	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA classes not included in the JDBC, Lib, Persist, or Kernel categories when extended trace is enabled.
openjpa.xtrace.ApiSpi	OFF, ALL, FINER, FINEST	Extended trace for public API/SPI interfaces defined for WsJPA, OpenJPA and JPA when extended trace is enabled.

3. Save the application server configuration and restart the application server.

Results

After you restart the application server, the new trace settings are picked up.

What to do next

Note: Tracing can degrade performance significantly and should be disabled when not in use. To disable trace, remove the Generic JVM argument and any trace detail levels added for enhanced tracing.

Enabling Enhanced Tracing for Java Persistence API (JPA) in a Java SE environment

In some cases it may be necessary to run a JPA trace from a Java SE environment.

About this task

Enabling enhanced tracing for a Java SE environment

Enhanced trace can also be used gather additional JPA trace information in a Java SE environment. In the WebSphere Application Server environment, the application server takes care of coupling the standard OpenJPA trace with the enhanced trace function to produce a single trace output. This must be done manually in a Java SE environment. Here are the steps to use enhanced tracing in a Java SE environment:

1. Create a logging configuration properties file. This file must use the standard java.util.logging configuration file format. Below is a sample configuration file. The trace categories defined in the table below can also be used in the configuration file. As standard for java.util.logging configuration files, trace categories must be suffixed with .level.

```
# Sample logger.properties file # Enable the file handler handlers =
 java.util.logging.FileHandler # Set a trace file pattern - this pattern will write a file named # jpa_jse.log into the current
 directory  java.util.logging.FileHandler.pattern = jpa_jse.log # Setup the basic logging level of the file handler
 java.util.logging.FileHandler.level = ALL # Set the standard OpenJPA jdbc.SQL category to trace level to ALL
 openjpa.jdbc.SQL.level = ALL # Set the enhanced OJPA_General category to trace at FINEST openjpa.OJPA_General.level = FINEST #
 Set the enhanced OJPA_Kernel category to trace at FINER openjpa.OJPA_Kernel.level = FINER
```

Category	Relevant trace levels	Description
JPA	OFF, ALL, FINER, FINEST	Adds extended trace to the JPA trace group.
openjpa.*	OFF, ALL, FINER, FINEST	Normal OpenJPA trace in addition to extended trace for all categories in OpenJPA when extended trace is enabled.
openjpa.xtrace.*	OFF, ALL, FINER, FINEST	Extended trace for all categories in OpenJPA when extended trace is enabled.
openjpa.xtrace.Jdbc	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA JDBC classes when extended trace is enabled.
openjpa.xtrace.Lib	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA library classes when extended trace is enabled.
openjpa.xtrace.Persist	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA persistence classes when extended trace is enabled.
openjpa.xtrace.Kernel	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA kernel classes when extended trace is enabled.
openjpa.xtrace.General	OFF, ALL, FINER, FINEST	Extended trace for OpenJPA classes not included in the JDBC, Lib, Persist, or Kernel categories when extended trace is enabled.

2. Modify your persistence.xml file to use Apache commons logging instead of the default OpenJPA logger. Add this property to your persistence unit:

```
<property name="openjpa.Log" value="commons"/>
```

3. Add the apache commons logging Java archive (JAR) file to your class path. You can download the JAR file from the Apache Web site.

4. Add the following argument to the Java virtual machine (JVM), where <WAS install path> is the fully qualified path of your application server install directory. Be sure to use the path separator character appropriate for your operating system. This parameter must be specified before the name of the class or JAR file to be executed.

```
-javaagent:<WAS install path>/optionalLibraries/IBM/wsjpa/wsjpatrace.jar
```

5. Add this additional argument to the JVM which specifies the path to your logging configuration file. This option must also be specified before the name of the class or jar to be executed.

```
-Djava.util.Logging.config.file=Logger.properties
```

6. Run your Java SE application. Here is a sample Java SE application invocation with enhanced trace enabled:

```
java -javaagent:"C:\Program
 Files\IBM\WebSphere\AppServer\optionalLibraries\IBM\wsjpa\wsjpatrace.jar" -Djava.util.logging.config.file=logger.properties
 my.JPAApplication
```

Results

Enhanced tracing is now functioning in your Java SE environment.

What to do next

Note: Use of extended trace agent should not be used in combination with the OpenJPA PCEnhancer agent in a Java SE environment. If enhanced tracing is used, the OpenJPA PCEnhancer must be used at compile time. If the enhancer and enhanced trace agents are used together the results are unpredictable.

Troubleshooting Java Persistence API (JPA) deadlocks and transaction time-outs

If an application encounters deadlocks and locked time-outs from the database, you can adjust the locking or transaction type to resolve the problem. Transactions are critical to maintaining data integrity. They are used to group operations into units of work that act together in an all-or-nothing fashion. There are two types of transactions, *optimistic* and *pessimistic*. JPA for WebSphere Application Server uses optimistic transactions by default, however, OpenJPA also supports pessimistic transactions. This topic includes the steps to configure these transactions.

Before you begin

Pessimistic transactions prevent all other concurrent transactions from using or modifying the same data by locking the datastore records that they are using. The advantage of a pessimistic transaction is that conflicts between multiple transactions are avoided. However, preventing these conflicts consumes more database resources than optimistic transactions. Because pessimistic transactions lock their records, it is possible that two transactions can wait for the other to release its lock before continuing. This results in a deadlock. The result is that one transaction is forced to unlock its records after a specified timeout interval and an exception is generated.

Optimistic transactions operate with the assumption that transactions do not conflict with each other. Optimistic transactions do not lock datastore records. Therefore, it is possible for two transactions to access and change the persistent information at the same time. Optimistic transactions consume less database resources than pessimistic transactions but they can be less reliable.

These changes are not detected until one transaction attempts to commit its changes. At that time, the transaction realizes that the data has been changed by another transaction and an OptimisticLockException exception displays. Despite this disadvantage, optimistic transactions are typically a better choice for applications because of performance and a low risk of deadlock.

Locking

Servers use objects called locks to maintain transaction and data integrity. Locks are managed automatically by lock management programs. Locking is handled on a per-transaction basis. When you obtain a lock on an entity, the lock indicates that you have the right to use the entity. The lock indicates that no other user can alter the data of the entity until the lock is released. Locking management ensures that no other user can obtain a lock that conflicts with a lock already in place. A lock cannot be used by more than one user. Locking management automatically assigns and releases locks, according the actions of the user.

To manage locks with JPA for WebSphere Application Server, you need to use the LockManager plug-in. The LockManager controls how the JPA application locks the data involved in the transaction. Depending on the setting used, LockManager determines if locking takes place, and if so, optimistic or pessimistic. In addition to LockManager, you need to control the level of isolation that the lock places on the entities and determine when the locking takes place in the transaction.

JPA for WebSphere Application Server contains advanced locking and versioning APIs for more control over database resources and object versions. The implementation works with the application server container for managed transactions. This allows the use of the declarative transaction demarcation and Java Transaction API (JTA) implementations within the container.

JPA Access Intent

JPA Access Intent in JPA for WebSphere Application Server provides an application improved control over the settings of isolation level and read lock when accessing data. This ability allows the data access criteria to be set based on the entity use and the taskName associated to the current transaction. By refining the scope of access, the deadlock condition is further minimized.

About this task

The following steps show you how to enable optimistic or pessimistic locking:

- Enable Optimistic locking by setting the isolation and LockManager levels: Optimistic locking should be used when the chance of contention between updating transactions is low or when you have to read and update transaction across multiple transactions.
  1. Ensure that the LockManager is set to the following version: openjpa.LockManager = ″version″. This is the default setting.
  2. Ensure that the isolation level is set to Read_Committed: openjpa.TransactionIsolation = ″Read_Committed″. This is the default setting.
  3. Place the @Version annotation column in the entity class definition.
  4. Set the LockMode API for the EntityManager to guarantee repeatable read on a per entity instance, if needed: EntityManager.setLock(LockMode.READ) or EntityManager.setLock(LockMode.WRITE).

  Optimistic locking is now in place with versions enabled to monitor the entities involved in transactions and repeatable read permitted in case of OptimisticLockException exceptions. If you still experience frequent OptimisticLockException exceptions because transactions are rolled back and reinitiated, you should use pessimistic locking.
- Enable pessimistic locking by modifying the LockManager and isolation levels:
  1. Change the LockManager from version to pessimistic: openjpa.LockManager=″pessimistic″
  2. Set the isolation level to Repeatable_Read: openjpa.TransactionIsolation=″Repeatable_Read″. For more information about this property, see the OpenJPA documentation.

  The locking type is now pessimistic. You might experience deadlocks if two transactions access read data, obtain read locks and attempt to update the data concurrently.
- If you encounter frequent deadlocks, you can modify your pessimistic locking as follows:
  1. Set the isolation level to Serializable: openjpa.TransactionIsolation=″Serializable″.

2. Modify the ReadLockLevel property to ″write″: openjpa.ReadLockLevel=″write″. You can also use query hint with this property: openjpa.FetchPlan.ReadLockLevel=″write″.

After your modifications are done, transactions acquire an update lock on data and cause other update transactions to serialize. The transaction throughput is less because transactions are serialized, but they execute without deadlock.

Results

For more information, consult chapter 9 of the JPA Reference guide.

Related tasks

JPA Access Intent
Java Persistence API (JPA) Access intent specifies the isolation level and lock level used when reading data from a data source. Access intent controls the JDBC isolation level and whether read, update or exclusive locks are acquired when retrieving data.

wsjpaversion command

Use this command line tool to find out information about the installed version of Java Persistence API (JPA) for WebSphere Application Server.

Syntax

The command syntax is as follows:

```
wsjpaversion
```

Usage

The version tool can be very useful when debugging problems with JPA in applications and for providing customer support teams with the information about the current JPA environment.

Examples

Find the version information of your JPA installation:

```
WSJPA 1.2.0-SNAPSHOT
version id: WSJPA-1.2.0-SNAPSHOT-r1118:1319
revision: 1118:1319

OpenJPA 1.2.0-SNAPSHOT
version id: openjpa-1.2.0-SNAPSHOT-r420667:634150
Apache svn revision: 420667:634150

os.name: Linux
os.version: 2.6.9-55.0.9.ELsmp
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
app_server_root\dev\JavaEE\5.0\j2ee.jar
app_server_root\plugins\com.ibm.ws.jpa.jar

user.dir: /opt/WebSphere70/AppServer9/bin
```

On Windows operating systems the output looks like this:

```
C:\a.was.src\WASX Iww0811.63 S11.48 F11.32>wsjpaversion
WSJPA 1.2.0-SNAPSHOT
version id: WSJPA-1.2.0-SNAPSHOT-r1118:1356
revision: 1118:1356
```

```
OpenJPA 1.2.0-SNAPSHOT
version id: openjpa-1.2.0-SNAPSHOT-r420667:638667M
Apache svn revision: 420667:638667M

os.name: Windows XP
os.version: 5.1 build 2600 Service Pack 2
os.arch: x86

java.version: 1.6.0
java.vendor: IBM Corporation

java.class.path:
app_server_root\dev\JavaEE\5.0\j2ee.jar
app_server_root\plugins\com.ibm.ws.jpa.jar
c:\a.tools.common\java.tools

user.dir: C:\a.was.src\WASX Iww0811.63 S11.48 F11.32

C:\a.was.src\WASX Iww0811.63 S11.48 F11.32>
```

Using object cache instances

Perform this task so that your application can access dynamic cache object cache instances with the DistributedMap or DistributedObjectCache interfaces.

Before you begin

Before you begin, enable the dynamic cache service. See “Using the dynamic cache service” on page 2210 for more information.

About this task

Using object cache instances can improve the performance of your application because you can programmatically store and share frequently used objects. By using object cache instances, you also have the necessary control over the dynamic cache when you are running multiple applications in an application server. See “Cache instances” on page 1336 for more information.

1. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Object cache instances**.
   b. Specify the scope for the cache instance.

      Node scope makes the cache instance available to all servers on the particular node. Server scope makes the cache instance available to only the selected server. You can mix scopes, if necessary.
   c. Click **Apply** after changing the scope.
   d. Click **New**.
   e. Enter the Java Naming and Directory Interface (JNDI) name for this cache instance.

      This is name that you pass to the InitialContext lookup() method from within your application. For example, `services/cache/instance_one`.
   f. Enter or modify other properties as needed.
2. Update your application. To store and retrieve objects in an object cache instance, you need a DistributedMap or DistributedObjectCache reference for the named object cache instance. See “Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache” on page 2253 for more information.

Results

You configured object cache instances that you can access programmatically with the DistributedMap and DistributedObjectCache interfaces.

Administering data access applications

These administrative tasks consist primarily of configuring the objects, or resources, through which applications connect with a backend, and tuning those resources to handle the volume of connection requests.

1. If your application contains Web modules or EJB modules that require access to a backend, configure resources according to your type of enterprise information system (EIS):
   - For a relational database, follow the steps outlined in the “Configuring a JDBC provider and data source” on page 1431 topic. If you are using a DB2 database, the ″Configuring an application to use pureQuery″ topic found in the related links section is another option. PureQuery provides an alternative to JDBC as a way to access the DB2 database.
   - For a non-relational database, or another type of EIS such as the Customer Information Control System (CICS), you must configure a resource adapter and connection factories. The topic Accessing data using Java EE Connector Architecture connectors provides information on setting up these objects.

   Note: When you specify the Java Naming and Directory Interface (JNDI) name for resources, adhere to the following requirements:
   - Do not assign duplicate JNDI names across different resource types (such as data sources versus J2C connection factories or JMS connection factories).
   - Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.
2. Configure an authentication alias for the new Web module resource or EJB module resource only if the application code, rather than WebSphere Application Server, authenticates connections with the backend. This security configuration is called component-managed authorization, and is indicated in the application deployment descriptor as `res-auth = Application`.

   Container-managed authorization, which is designated as `res-auth = Container`, indicates that Application Server performs signon for backend connections. The container-managed authentication alias must be specified on the application resource reference. This task can be done during application assembly or deployment, along with mapping the resource reference to a data source or connection factory resource. After application deployment, however, you can alter the container-managed authentication alias using the administrative console. Click **Applications** → **Websphere enterprise applications** → ***application_name***, and select the link to the appropriate mapping page. For example, if you want to alter the alias of an EJB module resource, you might click **Map data sources for all 2.x CMP beans**. For a Web module resource, click **Resource References**.

   Consult the “J2EE connector security” on page 1426 topic for detailed reference on resource authentication.
3. If your application contains a client module that requires data access, see Configuring data access for application clients. In this single configuration process, you can define authentication data for either component-managed or container-managed signon.
4. Specify connection pool settings.
5. Test a connection to the new data source. See the article “Test connection service” on page 1521 for information on the available methods for testing connections. This article also addresses important data source settings that can affect the accuracy of your test connection results.
6. Set the JDBC trace service. The JDBC trace log information augments the JVM log data for data source failures.

   To activate the trace using the administrative console, consult Enabling trace at server startup. Specify **WAS.database** as the trace group and select **com.ibm.ws.db2.logwriter** as the trace string.
7. Gather connection pool statistics by activating the JDBC connection pool counters or the J2C connection pool counters. Alternatively, you can use Performance Monitoring Infrastructure (PMI) method calls to gather connection statistics; consult the article “Connection and connection pool statistics” on page 1323.

8. Tune your database to accommodate the connection volume. If you use DB2 UDB for iSeries, consult the “DB2 Universal Database performance tips” on page 1544 article as a starting point reference.

Configuring Java EE Connector connection factories in the administrative console

To access an enterprise information system (EIS), configure connection factories, which instantiate resource adapter classes for establishing and maintaining resource connections.

About this task

An application component uses a connection factory to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS). Examples of connections include database connections, Java Message Service connections, and SAP R/3 connections.

1. Click **Resources** → **Resource Adapters** → **Resource adapters**.
2. In the **Resource adapters** panel, select the resource adapter that you want to configure.
3. From the **Additional Properties** heading, click **J2C connection factories**.
   a. Click **New**.
   b. Specify any properties for the connection factory in the **General Properties** panel.
   c. Select the authentication preference.
   d. Select the aliases for **Component-managed authentication**, **Container-managed authentication**, or both. Some choices for the mapping-configuration alias do not use a container-managed authentication alias, so you will not be able to select a container-managed alias if one of those mapping-configuration aliases is selected.

      If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

      Note: If the resource adapter supports XA, an option for **Authentication alias for XA recovery** will be available.

      If there are no aliases that are available, or you want to define a different alias:
      1) Click **Apply** to save the current settings.
      2) Click **JAAS - J2C authentication data** from the **Related Items** heading.
      3) Click **New**.
      4) Define the properties for the alias in **General Properties**.
      5) Click **OK**.
   e. Click **OK**.
4. Click the name of the J2C connection factory that you created.
5. From the **Additional Properties** heading, click **Connection pool properties**.
   a. Change any values by clicking the property name. For more information on the settings for connection pools, read “Tuning connection pools” on page 1462 or “Connection pool settings” on page 1400.
   a. Click **OK**.
6. Click **Custom properties** from the **Additional Properties** heading.
   a. Click any property name to change its value. If the **UserName** and **Password** properties are defined, they will be overridden by the component-managed authentication alias that you specified in the previous step.
   b. Click **Save**.

7. Restart the application server for the changes to take effect.

Configuring connection factories for resource adapters within applications

To access an enterprise information system (EIS), configure connection factories, which instantiate resource adapter classes for establishing and maintaining resource connections.

About this task

An application component uses a *connection factory* to access a connection instance, which the component then uses to connect to the underlying enterprise information system (EIS). Examples of connections include database connections, Java Message Service connections, and SAP R/3 connections.

1. Optional: Install the application if it is not already installed on the application server.
   a. Click **Applications** > **Install New Application**.
   b. Browse to find the appropriate EAR file, which contains an RAR file.
   c. Click **Next**.
   d. Complete the installation process for the application. For more information on installing applications, refer to Installing enterprise application files with the console.
2. Select the application that you want to configure.
3. Click **Modules** > **Manage Modules**.
4. Select the name of the RAR file in the **Manage Modules** panel.
5. Click **Resource Adapter** under the **Additional Properties** heading.
6. Under the **Additional Properties** heading, click **J2C connection factories**.
   a. Click **New**.
   b. Specify any properties for the connection factory in the **General Properties** panel.
   c. Select the authentication preference.
   d. Select an alias for **Component-managed authentication** if any application components with *Application* or *Per connection factory* authentication specified in the resource reference are going to be getting connections from this connection factory using the empty-argument getConnection() method. For resources that support XA, you can specify an Authentication alias for XA recovery. If there are no aliases that are available, or you want to define a different alias:
      1) Click **Apply** to save the current settings.
      2) Click **JAAS - J2C authentication data** under the **Related Items** heading.
      3) Click **New**.
      4) Define the properties for the alias in **General Properties**.
      5) Click **OK**.
   e. Click **OK**.
7. Click the name of the J2C connection factory that you created.
8. Under the **Additional Properties** heading, click **Connection pool properties**.
   a. Change any values by clicking on the property name. For more information on the settings for connection pools, refer to “Tuning connection pools” on page 1462 or “Connection pool settings” on page 1400.
   a. Click **OK**.
9. Click **Custom properties** under the **Additional Properties** heading.
   a. Click any property name to change its value. If the **UserName** and **Password** properties are defined, they will be overridden by a **component-managed authentication** alias that you might have configured.
   b. Click **Save**.

Connection pool settings

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types, like JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

For example: click **Resources** → **JDBC** → **JDBC Providers** → ***JDBC_provider*** → **Data Sources** → ***data_source*** → **Connection pool properties**

The path for JMS queue connection factories is:**Resources** → **JMS** → **Queue connection factories** → ***JMS_queue_connection_factory*** → **[Additional properties] Connection pool properties**.

Connection timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds that a connection request waits when there are no connections available in the free pool and no new connections can be created. This usually occurs because the maximum value of connections in the particular connection pool has been reached.

For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. In most cases you should not retry the getConnection() method; if a longer wait time is required you should increase the Connection Timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.

If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Maximum connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown. For example: If the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

Knowing the number of connection pools that can potentially request connections from the backend (such as a DB2 database or a CICS server) helps you determine a value for the Maximum Connections property.

For multiple standalone application servers that use the same data source configuration, or J2C connection factory configuration, a separate physical connection pool exists for each server. If you clone these same application servers, WebSphere Application Server implements a separate connection pool for each clone.

All of these connection pools correspond to the same data source or connection factory configuration. Therefore all of these connection pools can potentially request connections from the same backend resource, at the same time. The single Maximum Connections value that you set on this console panel applies to every one of these connection pools. Consequently, setting a high Maximum Connections value can result in a load of connection requests that overwhelms your backend resource.

Data type	Integer
Default	10
Range	0 to maximum integer If Max Connections is set to 0, the Connection Timeout value is ignored.

Note: For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum connections:

Specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example, if the Minimum Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting Aged Timeout to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached, the application server will not discard the connection until after the transaction is completed and the connection is closed.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**.

Data type	String
Defaults	• EntirePool for J2C connection factories and JMS-related connection factories • EntirePool for WebSphere Version 4.0 data sources • EntirePool for current version data sources that you create through the administrative console • EntirePool for current version data sources that you script through wsadmin AdminConfig commands, invoking JDBC templates that are built into WebSphere Application Server (For information on the command **createUsingTemplate**, see the information center article ″Commands for the AdminConfig object.″) • FailingConnectionOnly for data sources that you script in wsadmin without invoking JDBC templates :

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

Connection pool advanced settings

Use this page to specify connection pooling related settings.

This administrative console page is common to a range of resource types: for example, JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection pool properties** → **Advanced connection pool properties**.

For example, click:

- **Resources** → **JDBC** → **JDBC providers** → ***JDBC_provider*** → **Data sources** → ***data_source*** → **Connection pool properties** → **Advanced connection pool properties**
- **Resources** → **JMS** → **JMS provider** → **Default messaging** → **Queue connection factory** → ***JMS_queue_connection_factory*** → **Connection pool properties** → **Advanced connection pool properties**.

The number of shared partitions, the number of free pool partitions, and the free pool distribution table size are properties related to reducing the time a thread needs to wait for a synchronization lock. On systems with a single processor, these values make no difference. On systems with multiple processors, these settings can reduce the performance cost associated with managing multiple threads.

Related concepts

“Resource adapters” on page 1315
A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

“JDBC providers” on page 1319
Installed applications use JDBC providers to interact with relational databases.

Number of shared partitions:

Specifies the number of partitions that are created in each of the shared pools.

Partition support is always enabled. The default values of 0 should be used to enable the connection pool to pick the best values for performance. In some cases where large multiprocessor systems are used, adjusting the partition support properties might help performance.

Data type	integer
Default value	0
Range	0 to max int

Number of free pool partitions:

Specifies the number of partitions that are created in each of the free pools.

Data type	integer
Default value	0
Range	0 to max int

Free pool distribution table size:

Determines the distribution of Subject and CRI hash values in the table that indexes connection usage data.

These hash values are used to match connection request credentials with the connections. A free pool distribution table size larger than 1 can yield more efficient distribution of hash values, to help minimize search collisions within the table. Fewer collisions can result in faster retrieval of a connection that matches a request. Use a larger value for free pool distribution table size if your resource receives many incoming requests with varying credentials. Smaller values (1) should be used if the same credentials apply to all incoming requests for the resource. The value of 0 means random distribution.

Data type	integer
Default value	0
Range	0 to max int

Surge threshold:

Specifies the number of connections created before surge protection is activated.

Surge protection is designed to prevent overloading of a data source when too many connections are created at the same time. Surge protection is controlled by two properties, *surge threshold* and *surge creation interval*.

The surge threshold property specifies the number of connections created before surge protection is activated. After you reach the specified number of connections, you enter *surge mode*.

The surge creation interval property specifies the amount of time, in seconds, between the creation of connections when in surge mode.

For example, assume the follow settings:
- maxConnections = 50
- surgeThreshold = 10
- surgeCreationInterval = 30 seconds

If the connection pool receives 15 connection requests, 10 connections are created at about the same time. The 11th connection is created 30 seconds after the first 10 connections. The 12th connection is created 30 seconds after the 11th connection. Connections continue to be created every 30 seconds until there are no more new connections needed or you reach the maxConnections value.

Surge connection support starts if the surge threshold is > -1 and the surge creation interval is > 0. The surge threshold property has a default value of -1, which indicates that it is turned off.

wsadmin examples

```
$AdminControl getAttribute $objectname surgeCreationInterval
$AdminControl setAttribute $objectname surgeCreationInterval 30
$AdminControl getAttribute $objectname surgeThreshold
$AdminControl setAttribute $objectname surgeThreshold 15
```

Data type	integer
Default value	-1
Range	-1 to max int

Surge creation interval:

Specifies the amount of time between connection creates when you are in surge protection mode.

When the number of connections specified for the surge threshold property is reached, the surge creation interval property dictates how much time each new connection request must wait before fulfillment.

Note: Surge protection does not work for a connection pool that is managed through an activation specification that coordinates with a JMS queue connection factory and default messaging provider. To control incoming connections for JMS calls such as **onMessage**, refer to the help article for the administrative console page **JMS** > **Activation specification** > *activation_specification_name*.

Data type	integer
Default value	0
Range	0 to max int

Stuck timer interval:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. The stuck timer interval property is the interval for the timer. This is how often the connection pool checks for stuck connections. The default value is 0 seconds.

If an attempt to change the stuck time, stuck timer interval, or stuck threshold properties using the wsadmin scripting tool fails, an `IllegalState` exception occurs. The pool cannot have any active requests or active connections during this request. For the stuck connection support to start, the stuck time and the stuck threshold property values must be greater than 0 and maximum connections must be greater than 0.

Also, the stuck timer interval, if it is set, must be less than the stuck time value. In fact, it is suggested that the stuck timer interval should be one-quarter to one-sixth the value of stuck time so that the connection pool checks for stuck connections 4 to 6 times before a connection is declared stuck. This reduces the likelihood of false positives.

wsadmin examples

```
$AdminControl getAttribute $objectname stuckTime
$AdminControl setAttribute $objectname stuckTime 30
$AdminControl getAttribute $objectname stuckTimerInterval
$AdminControl setAttribute $objectname stuckTimerInterval 15
$AdminControl getAttribute $objectname stuckThreshold
$AdminControl setAttribute $objectname stuckThreshold 10
```

Data type	integer
Default value	0
Range	0 to max int

Stuck time:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. The stuck time property is the interval, in seconds, allowed for a single active connection to be in use to the backend resource before it is considered to be stuck.

Data type	integer
Default value	0
Range	0 to max int

Stuck threshold:

A *stuck* connection is an active connection that is not responding or returning to the connection pool. If the pool appears to be stuck (you have reached the stuck threshold), a resource exception is given to all new connection requests until the pool is unstuck. An application can explicitly catch this exception and continue processing. The pool will continue to periodically check for stuck connections when the number of stuck connections is past the threshold. If the number of stuck connections drops below the stuck threshold, the pool will detect this during its periodic checks and enable the pool to begin servicing requests again. The stuck threshold is the number of connections that need to be considered stuck for the pool to be in stuck mode.

Data type	integer
Default value	0
Range	0 to max int

Connection pool (Version 4) settings

Use this page to create a connection pool for a Version 4.0 data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data sources (WebSphere Application Server V4)** > *data_source* > **Connection pool properties (version 4)**
- **Resources** > **JDBC** > **Data sources (WebSphere Application Server V4)** > *data_source* > **Connection pool properties (version 4)**

Scope: Resources such as JDBC providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

Note that no matter what the scope of a defined resource, the resource's properties only apply at an individual server level. For example, if you define the scope of a data source at the cell level, all users in that cell can look up and use that data source, which is unique within that cell. However, resource property settings are local to each server in the cell. For example, if you define *max connections* to 10, then each server in that cell can have 10 connections.

When resources are created, they are always created into the current scope selected in the panel. To view resources in other scopes, specify a different node or server in the scope selection form.

For general information, see *Administrative console scope settings* in the Related Reference section.

Data type String

Minimum pool size:

Specifies the minimum number of connections to maintain in the pool.

The minimum pool size can affect the performance of an application. Smaller pools require less overhead when the demand is low because fewer connections are held open to the database. When the demand is high, the first applications experience a slow response because new connections are created if all others in the pool are in use.

Data type Integer
Default 1
Range Any non-negative integer.

Maximum pool size:

Specifies the maximum number of connections to maintain in the pool.

If the maximum number of connections is reached and all connections are in use, additional requests for a connection wait up to the number of seconds specified as the connection timeout. The maximum pool size can affect the performance of an application. Larger pools require more overhead when demand is high because there are more connections open to the database at peak demand. These connections persist until idled out of the pool. If the maximum value is smaller, longer wait times or possible connection timeout errors during peak times can occur. Ensure that the database can support the maximum number of connections in the application server, in addition to any load that it has outside of the application server.

Data type Integer
Default 10
Range Any positive integer

Connection timeout:

Specifies the maximum number of seconds an application waits for a connection from the pool before timing out and triggering a `ConnectionWaitTimeout` exception. WebSphere Application Server acts on this value only if you set the maximum pool size property, in which case the number of maximum connections serves as a trigger for enforcing the wait timeout property.

Data type Integer
Units Seconds
Default 180
Range Any non-negative integer

Setting this value to 0 disables the connection timeout.

If you accept the default value, Application Server issues the ResourceAllocation exception immediately after the pool manager indicates that the maximum number of connections are in use. If you disable connection timeout, Application Server does not issue an exception. Instead, the pool manager queues subsequent connection requests until it can allocate a connection.

Idle timeout:

Specifies the maximum number of seconds that an idle (unallocated) connection can remain in the pool before being removed to free resources.

Connections need to idle out of the pool because keeping connections open to the database can cause database memory problems. However, not all connections are idled out of the pool, even if they are older than the Idle Timeout setting. A connection is not idled if removing the connection would cause the pool to shrink below its minimum size. Setting this value to 0 disables the idle timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Orphan timeout:

Specifies the maximum number of seconds that an application can hold a connection without using it before the connection returns to the pool

If there is no activity on an allocated connection for longer than the Orphan Timeout setting, the connection is marked for orphaning. After another Orphan Timeout number of seconds, if the connection still has no activity, the connection returns to the pool. If the application tries to use the connection again, it is issued a stale connection exception. Connections that are enlisted in a transaction are not orphaned. Setting this value to 0 disables the orphan timeout.

Data type	Integer
Units	Seconds
Default	1800
Range	Any non-negative integer

Statement cache size:

Specifies the number of cached prepared statements to keep per connection.

The largest value you would need to set your cache size to if you do not want any cache discards is determined as follows: for each application that uses this data source on a particular server, add up the number of unique prepared statements (as determined by the *sql* string, concurrency, and the scroll type). This is the maximum number of possible prepared statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. This provides better performance. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	10
Range	Any non-negative integer

Disable auto connection cleanup:

Specifies whether the connection pooling software automatically closes connections from the data source at the end of a transaction. Set this property if you want to maintain and reuse the same connection across multiple transactions.

The default is *false*, which indicates that when a transaction completes, the application server closes the connection and returns it to the pool. Any use of the connection after the transaction has ended results in a stale connection exception, because the connection is closed and has returned to the pool. This mechanism ensures that connections are not held indefinitely by the application. If the value is set to true, the connection is not returned to the pool at the end of a transaction. In this case, the application must return the connection to the pool by calling the close() method. If the application does not close the connection, the pool can run out of connections for other applications to use.

Data type	Boolean (check box)
Default	False (clear)

J2C Connection Factories collection

Use this page to view Java 2 Connector (J2C) connection factories, which represent sets of connection configuration values.

Application components such as enterprise beans have resource reference descriptors that refer to the connection factory, not the resource adapter. The connection factory is really a configuration properties list holder. In addition to the arbitrary set of configuration properties defined by the vendor of the resource adapter, there are several standard configuration properties that apply to the connection factory. These standard properties are used by the Java 2 Connectors connection pool manager in the application server run time and are not known by the vendor-supplied resource adapter code.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Adapters** > **J2C connection factories**
- **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories**

Name:

Specifies a list of the connection factory display names.

Data type	String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name of this connection factory.

Data type	String

Scope:

Specifies the scope of the connection factory. Only applications that are installed within this scope can use this connection factory.

Provider:

Specifies the resource adapter that WebSphere Application Server uses for this connection factory.

Description:

Specifies a text description of this connection factory.

Data type String

Connection factory interface:

Specifies the fully qualified name of the interface that provides the implementation class for the connection factory.

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

J2C connection factories settings:

Use this panel to specify settings for a connection factory.

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **J2C connection factories** → ***J2C_connection_factory***
- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C connection factories** → ***J2C_connection_factory***

Scope:

Specifies the scope of the resource adapter that connects applications to an enterprise information system (EIS) through this connection factory. Only applications that are installed within this scope can use this connection factory.

Provider:

Specifies the resource adapter that WebSphere Application Server uses for this connection factory.

Provider is displayed in this location only when you create a new connection factory. The list shows all of the existing resource adapters that are defined at the relevant scope. Select one from the list if you want to use an existing resource adapter as Provider.

Create new provider:

Provides the option of configuring a new resource adapter for the new connection factory.

Create New Provider is displayed only when you create, rather than edit, a connection factory.

Clicking **Create New Provider** triggers the console to display the resource adapter configuration page, where you create a new adapter. After you click **OK** to save your settings, you see the connection factory collection page. Click **New** to define a new connection factory for use with the new resource adapter; the console now displays a configuration page that lists the resource adapter as the new connection factory Provider.

Name:

Specifies the name of this connection factory.

This is a required property.

Data type String

JNDI name:

Specifies the JNDI name of this connection factory.

For example, the name could be *eis/myECIConnection*.

After you set this value, save it and restart the server. You can see this string when you run the *dumpNameSpace* tool. This is a required property. If you do not specify a JNDI name, it is filled in by default using the Name field.

Data type String
Default eis/*display name*

Note: Adhere to the following requirements for JNDI names:

- Do not assign duplicate JNDI names across different resource types (such as data sources versus J2C connection factories or JMS connection factories).
- Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.

Description:

Specifies a text description of this connection factory.

Data type String

Connection factory interface:

Specifies the fully qualified name of the Connection Factory Interfaces supported by the resource adapter.

This is a required property. For new objects, the list of available classes is provided by the resource adapter in a drop-down list. After you create the connection factory, the field is a read only text field.

Data type Drop-down list or text

Category:

Specifies a string that you can use to classify or group this connection factory.

Data type String

Component-managed authentication alias:

Specifies authentication data for component-managed signon to the resource.

Select an alias from the list.

To define a new alias that is not displayed in the list:

- Click **Apply**. Under Related Items, you now see a listing for Java Platform, Enterprise Edition (Java EE) Connector Architecture (J2C) authentication data entries.
- Click **J2EE Connector Architecture (J2C) authentication data entries**.

- Click **New**.
- Define an alias.
- Click **OK**. The console now displays an alias collection page. This page contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
- Click the name of your J2C connection factory. You now see the configuration page for the connection factory.
- Select the new alias in the Component-managed authentication alias list.
- Click **Apply**.

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type List

The alias that you configure for component-managed authentication does not apply to all clients that must access the secured resource. External Java clients with Java Naming and Directory Interface (JNDI) access can look up a Java 2 Connector (J2C) resource such as a data source or Java Message Service (JMS) queue. However, they are not permitted to take advantage of the component-managed authentication alias defined on the resource. This alias is the default value that is used when the getConnection() method does not specify any authentication data, like *user* and *password*, or a value for ConnectionSpec. If an external client needs to get a connection, it must assume responsibility for the authentication by passing it through arguments on the getConnection() call.

Authentication alias for XA recovery:

This field is used to specify the authentication alias that should be used during XA recovery processing. If this alias name is changed after a server failure, the subsequent XA recovery processing will use the original setting that was in effect before the failure.

If the resource adapter does not support XA transactions, then this field will not be displayed. The default value will come from the selected alias for application authentication (if specified).

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Drop-down list

Mapping-configuration alias:

Specifies the authentication alias for the Java Authentication and Authorization Service (JAAS) mapping configuration that is used by this connection factory.

Click **Security** → **Global security** → **Java Authentication and Authorization Service** → **Application logins**, and select an alias from the table.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the userid and password. You may define and use other mapping configurations.

Note: Some mapping-configuration aliases do not use a container-managed authentication aliases, so you will not be able to select a container-managed authentication alias if one of those mapping-configuration aliases is selected.

Data type Pick-list

Container-managed authentication alias:

Specifies authentication data, which is a JAAS - J2C authentication data entry, for container-managed signon to the resource.

Select an alias from the list.

To define a new alias that is not displayed in the list:

1. Click **Apply**. Under Related Items, you now see a listing for Java Platform, Enterprise Edition (Java EE) Connector Architecture (J2C) authentication data entries.
2. Click **J2EE Connector Architecture (J2C) authentication data entries**.
3. Click **New**.
4. Define an alias.
5. Click **OK**. The console now displays an alias collection page. This panel contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
6. Click the name of your J2C connection factory. You now see the configuration panel for the connection factory.
7. Select the new alias in the container-managed authentication alias list.
8. Click **Apply**.

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Pick-list

Authentication preference (deprecated):

Specifies the authentication mechanisms defined for this connection factory.

Note: Beginning with WebSphere Application Server Version 6.0, the authentication preference is superseded by the combination of the <res-auth> application component deployment descriptor setting and the specification of a login configuration on the resource-reference mapping at deployment time.

This setting specifies which of the authentication mechanisms defined for the corresponding resource adapter applies to this connection factory. Common values, depending on the capabilities of the resource adapter, are: *KERBEROS*, *BASIC_PASSWORD*, and *None*.

If None is chosen, the application component is expected to manage authentication (<res-auth>Application</res-auth>). In this case, the user ID and password are taken from one of the following:

- The component-managed authentication alias
- UserName, Password Custom Properties

- Strings passed on the getConnection method

For example, if two authentication mechanism entries are defined for a resource adapter in the *ra.xml* document:
- <authentication-mechanism-type>BasicPassword</authentication-mechanism-type>
- <authentication-mechanism-type>Kerbv5</authentication-mechanism-type>

the authentication preference specifies the mechanism to use for container-managed authentication. An exception is issued during server startup if a mechanism that is not supported by the resource adapter is selected.

Data type	Pick-list
Default	BASIC_PASSWORD

J2C Connection Factory advanced settings:

Use this page to specify settings for a connection factory.

You can access this administrative console page in one of two ways:
- **Resources** > **Resource Adapters** > **J2C connection factories** > *J2C_connection_factory* > **Advanced connection factory properties**
- **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories** > *J2C_connection_factory* > **Advanced connection factory properties**

Related concepts

"Resource adapters" on page 1315
A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application.

"JDBC providers" on page 1319
Installed applications use JDBC providers to interact with relational databases.

Log missing transaction contexts:

Specifies whether or not the container logs that there is a missing transaction context when a connection is obtained.

Data type	Boolean
Default	True (enabled)

Cached handles:

Specifies whether cached handles (handles held in `inst vars` in a bean) should be tracked by the container.

Data type	Boolean
Default	False (clear)

Connection factory JNDI name practices

Observe the conventions of the Java Naming and Directory Interface (JNDI) service in WebSphere Application Server when you create connection factory JNDI names.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services use name-to-object mappings to associate

names with objects such locations, services, information, and resources. The Java Naming and Directory Interface (JNDI) provides a common interface that is used to access the various naming and directory services.

Naming your resources indirectly

When creating a connection factory or data source, a JNDI name is given by which the connection factory or data source can be looked up by a component. WebSphere Application Server uses an *indirect* name with the `java:comp/env` prefix:

- When you create a WebSphere Application Server data source, the default JNDI name is set to `jdbc/data_source_name`.
- When you create a connection factory, its default name is `eis/j2c_connection_factory_name`.

If you override these values by specifying your own, retain the `java:comp/env` prefix. An indirect name makes any resource-reference data associated with the application available to the connection management runtime, to better manage resources based on the res-auth, res-isolation-level, res-sharing-scope, and res-resolution-control settings.

Naming your resources for use with CMP

In addition, if you click the checkbox for the **Use this data source for container managed persistence (CMP)** option when you create the data source, another reference is created with the name of `eis/jndi_name_of_datasource_CMP`. For example, if a data source has a JNDI name of *jdbc/myDatasource*, the CMP JNDI name is *eis/jdbc/myDatasource_CMP*. This name is used internally by CMP and is provided simply for informational purposes.

Installing a resource adapter archive

The application server uses the classes and other code that comprise a resource adapter archive (RAR) to support the resource adapters that you configure.

Before you begin

A RAR file, which is often called a Java EE Connector Architecture (JCA) connector, must comply with the JCA Specification. Meet these requirements by using a supported assembly tool to assemble a collection of Java archive (JAR) files, other runnable components, utility classes, and so on, into a deployable resource adapter archive (RAR). You can then install the RAR file in the application server.

About this task

A resource adapter archive provides the classes and other code to support a resource adapter for access to a specific enterprise information system (EIS), such as the Customer Information Control System (CICS). Therefore, you can only configure resource adapters for an EIS after you install the appropriate RAR file.

1. Navigate to the **Resource adapter** panel. Click **Resources** → **Resource Adapters** → **Resource adapters**.
2. Install a new resource adapter archive.
   a. Click **Install RAR**. A dialog opens for installing a RAR file and configuring the associated resource adapter. Only click **New** if you want to configure a new resource adapter for a previously installed RAR file.

      Note: When installing a RAR file using this dialog, the scope you define on the **Resource adapters** panel has no effect on where the RAR file is installed. You can install RAR files only at the *node* level, which you specify on the **Install RAR** panel.
   b. Browse to find the appropriate RAR file.

- If your RAR file is located on your local workstation, select **Local path**, and browse to find the file.
- If your RAR file is located on your server, select **Server path**, and specify the fully qualified path to the file.

c. Click **Next**.

3. Enter the resource adapter name and any other properties needed under *General Properties*. For more details on the settings that you can configure, see the topic on configuring a resource adapter in the administrative console.
4. Click **OK**.

Results

You have installed a resource adapter archive that will provide access to the EIS when it is properly configured. If you need to configure more settings, or change some settings that were configured during the installation process, refer to the topic on configuring a resource adapter in the administrative console for more information.

Installing resource adapters within applications

Install resource adapters in your applications so they can access outside data sources.

1. Assemble an application with resource adapter archive (RAR) modules in it. See Assembling applications.
2. Install the application following the steps in Installing a new application. In the **Map modules to servers** step, specify target servers or clusters for each RAR file. Be sure to map all other modules that use the resource adapters defined in the RAR modules to the same targets. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (`plugin-cfg.xml`) for each Web server is generated based on the applications that are routed through it.

   Note: When installing a RAR file onto a server, WebSphere Application Server looks for the manifest (MANIFEST.MF) for the connector module. It looks first in the *connectorModule.jar* file for the RAR file and loads the manifest from the *_connectorModule.jar* file. If the class path entry is in the manifest from the *connectorModule.jar* file, then the RAR uses that class path.

   To ensure that the installed connector module finds the classes and resources that it needs, check the **Class path** setting for the RAR using the console. For more information, see "Resource adapter settings" on page 1422 and "WebSphere relational resource adapter settings" on page 1316
3. Click **Finish** > **Save** to save the changes.
4. Create connection factories for the newly installed application.
   a. Open the administrative console.
   b. Click **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C connection factories**.
   c. Click **New** to create a new connection factory, or click on an existing connection factory to update it.

   After you create and save the connection factories, you can modify the resource references defined in various modules of the application and specify the Java Naming and Directory Interface (JNDI) names of the connection factories wherever appropriate.

   Note: A given native library can only be loaded one time for each instance of the Java virtual machine (JVM). Because each application has its own classloader, separate applications with embedded RAR files cannot both use the same native library. The second application receives an exception when it tries to load the library.

If any application deployed on the application server uses an embedded RAR file that includes native path elements, then you must always ensure that you shut down the application server cleanly, with no outstanding transactions. If the application server does not shut down cleanly it performs *recovery* upon server restart and loads any required RAR files and native libraries. On completion of recovery, do not attempt any application-related work. Shut down the server and restart it. No further recovery is attempted by the application server on this restart, and normal application processing can proceed.

Install RAR

Use this page to install a RAR file in one of two ways. You can either upload a RAR file from the local file system, or specify an existing RAR file on a server. The RAR file must be installed at the node level, and you can select the node below.

For information on installing a resource adapter, see the article ″Installing a Resource Adapter Archive (RAR) file.″

Related tasks

“Installing a resource adapter archive” on page 1416
The application server uses the classes and other code that comprise a resource adapter archive (RAR) to support the resource adapters that you configure.

Related reference

Administrative console buttons
This page describes the button choices that are available on various pages of the administrative console, depending on which product features you enable.

Administrative console page features
This topic provides information about the basic elements of an administrative console page, such as the various tabs.

Administrative console preference settings
Use the preference settings to specify how you want information to display on an administrative console panel. The preference settings vary from one administrative console panel to another.

Local path:

Specifies the local path of the RAR.

Data type String

Server path:

Specifies the server path where the RAR is located.

Data type String

Scope:

Specifies the scope of the resource adapter. Only applications that are installed within this scope can use this adapter.

Configuring resource adapters

You can view a list of installed and configured resource adapters in the administrative console, as well as use the administrative console to install new resource adapters, create additional configurations of installed resource adapters, or delete resource adapter configurations. You can also configure a single instance resource adapter.

Before you begin

A resource adapter is an implementation of the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Specification that provides access for applications to resources outside of the server or provides access for an enterprise information system (EIS) to applications on the server. It can provide application access to resources such as DB2, CICS, SAP and PeopleSoft. It can provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of .rar. A resource adapter can be provided as a standalone adapter or as part of an application, in which case it is referred to as an embedded adapter.

About this task

Use this task to configure a standalone resource adapter archive file. Embedded adapters are installed as part of the application installation. This panel can be used to work with either kind of adapter.

1. Open the administrative console.
2. Select **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter***.
3. Set the scope setting. This field specifies the level to which this resource definition is visible. For general information, see the topic on administrative console scope settings in the Related Reference section. The Scope field is a read-only string field that shows where the particular definition for a resource adapter is located. This is set either when the resource adapter is installed, which can only be at the node level, or when a new resource adapter definition is added.
4. Configure the description. This field specifies a text description of the resource adapter. Use a free-form text string to describe the resource adapter and its purpose.
5. Set the archive path. Use this field to specify the path to the RAR file containing the module for this resource adapter. This property is required.
6. Set the classpath. The class path includes the list of paths or JAR file names that, together, form the location for the resource adapter classes. This list includes any additional libraries needed by the resource adapter. The resource adapter code base itself is automatically added to the class path, but you can use this field to specify anything that is needed and is not within the RAR.
7. Set the native path. The list of paths which forms the location for the resource adapter native libraries is set here. The resource adapter code base itself is automatically added to the class path, but if anything outside the RAR is needed it can be specified here.
8. Set the ThreadPool alias. The name of a thread pool that is configured in the server that is used by the resource adapter's Work Manager is specified in this field. If there is no thread pool configured in the server with this name, the default configured thread pool instance, named Default, is used. This property is only necessary if this resource adapter uses Work Manager.
9. Optional: Isolate the class loading for the resource adapter.

   Note: You can isolate a resource adapter to allow different versions of the same resource provider to be loaded in the same Java Virtual Machine (JVM). For example, you might want to deploy multiple applications on a single server, and each application requires different versions or implementations of the same resource adapter. You can now isolate each version or implementation of the resource adapter so that the classes of each adapter will be loaded by its own class loader, and the class loader will not inadvertently link with classes of the other versions or implementations of the adapter.

   a. Select **Isolate this resource provider**.

      Note: Be aware of the following conditions:

      - You cannot isolate a resource adapter if you specify a native library path. If you specify a native path manually, using wsadmin or editing the XML descriptors, the native paths are ignored and Application Server will issue a warning at run time. The Application

Server will define a value for the native library path for some JDBC providers; this behavior is intended to help you configure your provider when a native library path is necessary. If you do not require the native library path, delete the value, and you will be able to select the option to isolate the resource provider.

- If you are running a mixed cell environment, the application server will remove any isolated JDBC providers from nodes that are running at versions earlier than 7.0 if the provider is scoped for a version 7.0 cell, and you have not migrated the provider from an older release. If you want to use isolated resources at the cell level, do not use the resources in nodes that are running at versions earlier than 7.0. Define a resource at the node level, or avoid using the resource in nodes that are earlier than version 7.0, because there will be a ″Naming not found″ exception when the application server attempts to perform a lookup on an isolated resource at the cell level.

There are other general considerations that you should take into account when isolating any type of resource provider. Refer to the topic on considerations for isolated resource providers for more information.

b. Give the resource adapter a unique class path that is appropriate for that version.

10. Optional: Restrict the JVM to allow only one instance of the resource adapter. This setting prevents more than one instance of a resource adapter enterprise bean with a unique resource adapter implementation class name from existing in the same Java Virtual Machine (JVM). Enabling this setting imposes a highly restrictive environment on the system and should be used with caution. For example, if two applications use the same embedded resource adapter, only the first application to start will be able to access resources through its embedded resource adapter. If a standalone resource adapter is configured for a single instance, no applications that embed that same resource adapter will be able to access resources.
    a. Click **Advanced resource adapter properties**.
    b. Select **Restrict the JVM to allow only one instance of the resource adapter.**
11. Optional: Register this resource adapter with the high availability manager.

    Note: Registering the resource adapter with the high availability manager specifies that the high availability (HA) manager will manage the lifecycle of a JCA 1.5 resource adapter in a cluster, ensuring that applications using resource adapters for inbound communication remain highly available. To that end, appropriate use of the HA capability options enable you to set up an environment that will be able to implement failover for inbound activity when a server goes down.

    Do not select this option without first consulting the product documentation for the resource adapter, because this option requires the resource adapter to support high availability of inbound messaging. This field is only available on resource archives that allow definitions for activation specifications.
    a. Click **Advanced resource adapter properties**.
    b. Select **Register this resource adapter with the high availability manager**. This setting can be implemented with:
        - **Endpoint failover:** allows only one resource adapter in an HA group to receive messages across multiple servers. The result is that only one resource adapter can have endpoints active at one time.
        - **Resource adapter instance failover:** allows only one resource adapter in an HA group to be started across multiple servers. Inbound or outbound communication is limited to one resource adapter in the cluster, and the result is that only one runtime resource adapter instance can exist at one time.

Resource adapters collection

Use this panel to view the list of installed and configured resource adapters that you can use, install new resource adapters, create additional configurations of installed resource adapters, or delete resource adapter configurations. You can install a stand-alone resource adapter archive (RAR) file, or manage embedded adapters that are installed as part of the installation of an application.

You can configure a single instance resource adapter after a resource adapter archive (RAR) file is installed. The RAR file can either be stand-alone or embedded in an application through the administrative console or through the scripting tool. A checkbox is located on the console for you to specify that you want a single instance to be created at run time.

To view this administrative console page, click **Resources** → **Resource Adapters** → **Resource adapters**.

A resource adapter is an implementation of the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification that provides access for applications to resources outside of the server or provides access for an enterprise information system (EIS) to applications on the server. Resource adapters can provide application access to resources such as DB2, CICS, SAP and PeopleSoft. A resource adapter can provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of `.rar`. A resource adapter can be provided as a stand-alone adapter or as part of an application, in which case the resource adapter is referred to as an embedded adapter.

To view the resource adapters that are provided with the application server, select **Show built-in resources** in **Preferences**.

Scope:

Specifies the level at which this resource adapter is visible. For general information, read about administrative console scope settings.

Some considerations that you should keep in mind for this particular panel are:

- Changing the scope enables you to see which resource adapter definitions exist at that level.
- Changing the scope does not have any effect on installation. Installations are always done under a scope of node, no matter what you set the scope to.
- When you create a new resource adapter from this panel, you must change the scope to what you want it to be before clicking New.

Install RAR:

Specifies to install a resource archive (RAR). You can upload a RAR file from the local file system, or specify an existing RAR file on a server.

The RAR file must be installed at the node level.

New:

Specifies to create a copy of a resource archive that is already installed in the application server. This will create a copy of the resource adapter that you select in the table.

If you wish to create a copy of an installed resource adapter, specify a server for the scope, and click **New**. You cannot create a copy of a resource adapter at the node scope. If you want to install a new resource adapter, click **Install RAR**.

Delete:

Specifies to delete the copy of a resource adapter. This will delete the copy that you select in the table.

Update RAR:

Specifies to update the resource adapter that you select in the table. Update a resource adapter archive (RAR) file when you determine that a resource adapter, or a set of resource adapters, needs to be updated with a different version or implementation.

Different versions or implementations of resource adapters can include different settings, so updating your adapter might be beneficial if you require a certain set of configuration options. You can choose if you would like to update the resource adapter for all of the nodes in a cell or all the nodes in a cluster. If some of your nodes are earlier than Version 7.0, the RAR update will not be supported until those nodes are migrated to Version 7.0.

Name:

Specifies the name of the resource adapter.

Resource adapter settings:

Use this page to specify settings for a resource adapter.

A resource adapter is an implementation of the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification that provides access for applications to resources outside of the server, provides access for applications to an enterprise information system (EIS), or provides access for an EIS to applications on the server. Resource adapters provide applications access to resources such as DB2, CICS, SAP and PeopleSoft. Resource adapters can provide an EIS with the ability to communicate with message driven beans that are configured on the server. Some resource adapters are provided by IBM; however, third party vendors can provide their own resource adapters. A resource adapter implementation is provided in a resource adapter archive file (RAR); this file has an extension of `.rar`. A resource adapter can be provided as a stand alone adapter or as part of an application, in which case the resource adapter is referred to as an embedded adapter. Use this panel to install a stand alone resource adapter archive file. Embedded adapters are installed as part of the application installation.

To view this administrative console page, click one of the following paths:

- **Resources** → **Resource Adapters** → **Resource adapters** → **New**.
- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter***.
- Install a new resource adapter archive:
  1. Click **Resources** → **Resource Adapters** → **Resource adapters** → **Install RAR**.
  2. Specify a full path for the local file system or remote file system, and click **Next**.

Scope:

Specifies the highest topological level at which application servers can use this adapter.

The Scope field is a read-only string field that specifies where the particular definition for a resource adapter is located. The Scope field is set when the resource adapter is installed, which can only be at the node level, or when a new resource adapter definition is added.

Name:

Specifies the name of the resource adapter definition.

This property is a required string containing no spaces that is a meaningful text identifier for the resource adapter.

Description:

Specifies a text description of the resource adapter.

This description is a free-form text string to describe the resource adapter and its purpose.

Archive path:

Specifies the path to the installed resource archive file that contains the module for this resource adapter.

You can only select RAR files that are installed on the nodes within the selected scope, preventing you from configuring a selection that might fail for some of your nodes.

Note: For resources at the cell scope, the RAR files that are available are those that are installed on each individual node in the entire cell. For resources at a cluster scope, the RAR files that are available are those that are installed on each individual node in that particular cluster.

This property is required.

Data type String

Class path:

Specifies a list of paths or Java archive file (JAR) names that together form the location for the resource adapter classes.

Class path entries are separated by using the ENTER key and must not contain path separator characters like ';' or ':'. Class paths can contain variable (symbolic) names that can be substituted using a variable map. Check your driver installation notes for specific JAR file names that are required.

Native library path:

Specifies an optional path to any native libraries, which are .dll or .so files.

Native path entries are separated by using the ENTER key and must not contain path separator characters like ';' or ':'. Native paths can contain variable (symbolic) names that can be substituted using a variable map.

Isolate this resource provider:

Specifies that this resource provider will be loaded in its own class loader. This allows different versions of the same resource provider to be loaded in the same Java Virtual Machine. Give each version of the resource provider a unique class path that is appropriate for that version.

Ensure that all copies of a resource adapter have the same value for this option. For example, if you create a resource adapter at the cluster scope, the value of this option will be taken from the resource adapter archive (RAR) that you copy. When you create the copy, you cannot modify the value for any instances of that RAR, which would be the copies at the node or cluster scope in this example. If you need to modify the value, you have to delete the copies of the RAR until there is only one instance of that particular RAR that is left.

Note: You cannot isolate a resource provider if you specify a native library path.

Thread pool alias:

Specifies the name of a thread pool that is part of the server configuration for this resource adapter. Set this property only if the resource adapter uses the work manager service.

If you input a thread pool name that does not exist in the server configuration, the application server uses the name DEFAULT.

Advanced resource adapter properties:

Use this page to specify advanced settings for resource adapters that comply with the Version 1.5 Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) specification.

A resource adapter is an implementation of the Java EE Connector Architecture (JCA) specification that provides access for applications to an enterprise information system (EIS), like DB2, CICS, SAP and PeopleSoft, or provides access for an EIS to applications on the server. A resource adapter can also provide an EIS with the ability to communicate with message-driven beans that are configured on the server. Some resource adapters are provided by IBM, but third party vendors can provide their own resource adapters.

A resource adapter implementation is provided in a resource adapter archive file; this file has an extension of .rar. A resource adapter can be provided as a stand-alone adapter or as part of an application, in which case it is referred to as an embedded adapter. Use this panel to install a stand-alone resource adapter archive file. Embedded adapters are installed as part of the application installation.

To view this administrative console page, click **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **Advanced resource adapter properties**.

Restrict the JVM to allow only one instance of this resource adapter:

Prevents more than one instance of a resource adapter JavaBean with a unique resource adapter implementation class name from existing in the same Java Virtual Machine (JVM). This field is only available on resource archives that allow definitions for activation specifications.

Note: Enabling this setting imposes a restrictive condition on the inbound communications. For example, if two applications embed the same resource adapter, only the first application to start will be able to access resources through its embedded resource adapter. If a stand-alone resource adapter is configured for a single instance, no applications that embed that same resource adapter will be able to access resources.

Data type	Boolean (checkbox)
Default	False (disabled)

Updating a resource adapter archive

Use the resource adapter archive (RAR) update wizard to update RAR files to a newer version. The application server uses the classes and other code that comprise a resource adapter archive to support the resource adapters that you configure.

Before you begin

A resource adapter must be installed on the application server, and you must have a new version of the resource adapter that is compatible with the old version. You can create the new RAR file with an assembly tool, or the vendor for the resource adapter can provide the new version. If the new version of a RAR file is not compatible with the old version of the resource adapter, an update is not possible.

About this task

Update a RAR file when you determine that a resource adapter or a set of resource adapters needs to be updated with a different version. Different versions of resource adapters can include different settings, so updating your adapter might be beneficial if you require a certain set of configuration options. You can choose if you would like to update the resource adapter for all of the nodes in a cell or all the nodes in a cluster. If some of your nodes are earlier than Version 7.0, the RAR update will not be supported until those nodes are migrated to Version 7.0.

If you prefer to have more than one version of a resource adapter active in a given Java Virtual Machine (JVM), the update wizard will not provide the option of creating a new version and keeping the old. In this case, you will need to create an isolated resource adapter and configure it accordingly. Refer to the topic on configuring a resource adapter for more information.

1. Save all configuration changes.
2. Backup your configuration settings with the backupConfig tool. The backupConfig tool is located in the `app_server_root/bin` directory. Read the topic on the backupConfig command for more information on how to use this command.
3. Stop any servers that contain the RAR file that you will update. Refer to Administering applications and their environment for information on how to stop or start an application server.
4. Click **Resources** → **Resource Adapters**.
5. Select the checkbox next to the RAR file to update, and click **Update RAR**.
6. Specify the installation path for the RAR file, and click **Next**.
   - If your RAR file is located on the same workstation as your browser, select **Local file system**, and browse to find the file.
   - If your RAR file is located on the server workstation where the application server is installed, select **Remote file system**, and specify the fully qualified path to the file.
7. Review the configuration information that is provided for the RAR file. The following information is displayed for the RAR file:
   - Name
   - Current RAR version
   - New RAR version
   - Scope
   - Any existing copies of the resource adapter. The resource adapters with an asterisk (*) are copies of the resource adapter and must also be updated at the same time.

   Click **Next** when you are finished.
8. Optional: Edit any properties that were added by the new version of the resource adapter. You can also edit these properties after completing the update.
   a. Select a resource in the list to edit the new properties. Only resources with new properties are included in the list.
   b. Edit the resource properties. Use the table that is provided to set the values for the selected resource's new properties.
      - Select the **Set for all** checkbox to apply the property value to all of the resources of the same type.
      - Click the **Reset to Default** button to reset all of the properties to the default values that are defined in the RAR file. This only affects the selected resource.
   c. Click **Next**.
9. Review the summary panel, and click **Finish** when you are satisfied with the configuration settings. When you click Finish, all of the configuration changes will be automatically saved. To revert to an older version of the resource adapter you must perform the update process again, and specify the older version of the RAR file.

What to do next

If you are not satisfied with the results, and you backed up your configuration with backupConfig tool, use the restoreConfig tool to restore your backup configuration. Read the topic on the restoreConfig command for more information on how to use this command.

J2EE connector security

The Java 2 Platform, Enterprise Edition (J2EE) connector architecture defines a standard architecture for connecting J2EE to heterogeneous enterprise information systems (EIS). Examples of EIS include Enterprise Resource Planning (ERP), mainframe transaction processing (TP) and database systems.

The connector architecture enables an EIS vendor to provide a standard resource adapter for its EIS. A resource adapter is a system-level software driver that is used by a Java application to connect to an EIS. The resource adapter plugs into an application server and provides connectivity between the EIS, the application server, and the enterprise application. Accessing information in EIS typically requires access control to prevent unauthorized accesses. J2EE applications must authenticate to the EIS to open a connection.

The J2EE connector security architecture is designed to extend the end-to-end security model for J2EE-based applications to include integration with EIS environments. An application server and an EIS collaborate to ensure the correct authentication of a resource principal, which establishes a connection to an underlying EIS. The connector architecture identifies the following mechanisms as the commonly supported authentication mechanisms, although other mechanisms can be defined:

- BasicPassword: Basic user-password-based authentication mechanism that is specific to an EIS
- Kerbv5: Kerberos Version 5-based authentication mechanism

Applications define whether to use application-managed sign-on or container-managed sign-on in the resource-ref elements in the deployment descriptor. Each resource-ref element describes a single connection factory reference binding. The res-auth element in a resource-ref element, whose value is either `Application` or `Container`, indicates whether the enterprise bean code can perform sign-on or whether application server can sign-on to the resource manager using the principal mapping configuration. The resource-ref element is typically defined at application assembly time with an assembly tool. The `resource-ref` can also be defined, or redefined, at deployment time.

Application managed sign-on

To access an EIS system, applications locate a connection factory from the Java Naming and Directory Interface (JNDI) namespace and invoke the getConnection method on that connection factory object. The getConnection method might require a user ID and password argument. A J2EE application can pass in a user ID and password to the getConnection method, which subsequently passes the information to the resource adapter. Specifying a user ID and password in the application code might compromise some security, however.

The user ID and password, if coded into the Java source code, are available to developers and testers in the organization. Also, the user ID and password are visible to users if they decompile the Java class.

The user ID and password cannot be changed without first requiring a code change. Alternatively, application code might retrieve sets of user IDs and passwords from persistent storage or from an external service. This approach requires that IT administrators configure and manage a user ID and password using the application-specific mechanism.

To access this authentication data, the application server supports a component-managed authentication alias to be specified on a resource. This authentication data is common to all references to the resource. Click **Resources > Resource Adapters > J2C connection factories** > *configuration_name* . Select **Use component-managed authentication alias**.

When `res-auth=Application`, the authentication data is taken from the following elements, in order:

1. The user ID and password that are passed to the getConnection method
2. The component-managed authentication alias in the connection factory or the data source
3. The custom properties user name and password in the data source

The user name and password properties can be initially defined in the resource adapter archive (RAR) file.

Do not use the custom properties, which enable users to connect to the resources.

Container-managed sign-on

The user ID and password for the target enterprise information systems (EIS) can be supplied by the application server. The product provides container-managed sign-on functionality. The application server locates the proper authentication data for the target EIS to enable the client to establish a connection. Application code does not have to provide a user ID and password in the getConnection call when it is configured to use container-managed sign-on, and authentication data does not have to be common to all references to a resource. The uses a Java Authentication and Authorization Service (JAAS) pluggable authentication mechanism to use a pre-configured JAAS login configuration, and LoginModule to map a client security identity and credentials on the running thread to a pre-configured user ID and password.

The product supports a default many-to-one credential mapping LoginModule module that maps any client identity on the running thread to a preconfigured user ID and password for a specified target EIS. The default mapping module is a special purpose JAAS LoginModule module that returns a PasswordCredential credential that is specified by the configured Java 2 connector (J2C) authentication data entry. The default mapping LoginModule module performs a table lookup, but does not perform actual authentication. The user ID and password are stored together with an alias in the J2C authentication data list.

The J2C authentication data list is located on the Global security panel from **Java Authentication and Authorization Service > J2C Authentication data**. The default principal and credential mapping function is defined by the DefaultPrincipalMapping application JAAS login configuration.

J2C authentication data that is modified using the administrative console takes effect when the modification is saved into the repository, and Test Connection is performed. Also, J2C authentication data that is modified using wsadmin scripting takes effect when any application is started or restarted for a given the application server server process. J2C authentication data modification takes effect by invoking the SecurityAdmin MBean method, updateAuthDataCfg. Set the HashMap parameter to null to enable the Securityadmin MBean to refresh the J2C authentication data using the latest values in the repository.

Do not modify the DefaultPrincipalMapping login configuration because the product includes performance enhancements to this frequently used default mapping configuration. The product does not support modifying the DefaultPrincipalMapping configuration, changing the default LoginModule module, or stacking a custom LoginModule module in the configuration.

For most systems, the default method with a many-to-one mapping is sufficient. However, the product does support custom principal and credential mapping configurations. Custom mapping modules can be added to the application logins JAAS configuration by creating a new JAAS login configuration with a unique name. For example, a custom mapping module can provide one-to-one mapping or Kerberos functionality.

Trusted connections also provide a one-to-one mapping while supporting client identity propagation. In addition by utilizing the DB2 trusted context object, trusted connections can take advantage of connection pooling to reduce the performance penalty of closing and reopening connections with a different identity. Using trusted connections also enhances the security of your DB2 database by eliminating the need to assign all privileges to a single user. The connection is established by a user whose credentials are trusted by the DB2 server to open the connection and the same user is also then trusted to assert the identity of the other users accessing the DB2 server from the application. A new mapping configuration called TrustedConnectionMapping has been created to implemented trusted connections.

You also can use the WebSphere Application Server administrative console to bind the resource manager connection factory references to one of the configured resource factories. If the value of the res-auth

element is `Container` within the deployment descriptor for your application, you must specify the mapping configuration. To specify the mapping configuration, use the **Resource references** link under References on the **Applications** > **Application Types** > **WebSphere enterprise applications** > *application_name* panel. See Mapping resource references to references for additional directions.

J2C mapping modules and mapping properties

Mapping modules are special JAAS login modules that provide principal and credential mapping functionality. You can define and configure custom mapping modules using the administrative console.

You also can define and pass context data to mapping modules by using login options in each JAAS login configuration. In the product, you also can define context data using mapping properties on each connection factory reference binding.

Login options that are defined for each JAAS login configuration are shared among all resources that use the same JAAS login configuration and mapping modules. Mapping properties that are defined for each connection factory reference binding are used exclusively by that resource reference.

Consider a usage scenario where an external mapping service is used.

For example, you might use the Tivoli Access Manager global sign-on (GSO) service. Use the Tivoli Access Manager GSO to locate authentication data for both backend servers.

You have two EIS servers: DB2 and MQ. The authentication data for DB2 is different from that for MQ, however. Use the login option in a mapping JAAS login configuration to specify the parameters that are required to establish a connection to the Tivoli Access Manager GSO service. Use the mapping properties in a connection factory reference binding to specify which EIS server requires the user ID and password.

For more detailed information about developing a mapping module, see the J2C principal mapping modules article.

Note:

- The mapping configuration at the connection factory has moved to the resource manager connection factory reference. The mapping login modules that are developed using WebSphere Application Server Version 5 JAAS callback types can be used by the resource manager connection factory reference, but the mapping login modules cannot take advantage of the custom mapping properties feature.
- Connection factory reference binding supports mapping properties, and passes those properties to mapping login modules by way of a new WSMappingPropertiesCallback callback. In addition, the WSMappingPropertiesCallback callback and the new WSManagedConnectionFactoryCallback callback are defined in the com.ibm.wsspi package. Use the new mapping login modules with the new callback types.

Mapping resource references to references

You can use the administrative console to bind the resource manager connection factory references to one of the configured resource factories.

About this task

If the value of the res-auth element is `Container` within the deployment descriptor for your application, then you must specify the mapping configuration.

1. Click **Applications** > **Enterprise applications >** *application_name*.
2. From Additional Properties, select **Resource references**.

3. Select the application module and specify an authentication method for the selected connection factory reference binding. Select either **Use default method**, **Use custom login configuration** or **Use trusted connections**. If you select the **Use default method** option, the DefaultPrincipalMapping login configuration is selected. If you select the **Use trusted connections** option, then the TrustedConnectionMapping login configuration is selected. You must select an authentication data alias from the list.
4. After you make a selection, click **Apply** for the configuration to take effect.
5. If you select the **Use trusted connection option**, then you must select an authentication data alias from the menu list. The alias that is specified is what the application server uses to get the initial trusted connection.
6. Click **Apply**. The selected login configuration name and an **Mapping properties** button is displayed in the Login configuration field of the particular connection factory reference binding.
7. Click **Mapping properties > New** to specify the properties for your configuration. Click **OK** after specifying the properties on the Mapping properties panel.
8. If you select the **Use trusted connection** option, then you must select an authentication data alias from the menu list.
9. Click **Apply**. The **Mapping properties** button is displayed in the login configuration field of the particular connection factory reference binding.
10. Click **Mapping properties** to modify the properties of the trusted connection. Read about "Setting the security properties for trusted connections" on page 1503 for information on tuning the mapping properties for the trusted connection.
11. Click **OK** and **Save** on the Resource references panel to save your changes to the master configuration.

Managing messages with message endpoints

Manage message delivery for message-driven beans (MDB) that are deployed as message endpoints. The message endpoints are managed beans (MBeans) for inbound resource adapters that are compliant with Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Version 1.5.

About this task

The application server provides message endpoint MBeans to assist you in managing the delivery of a message to your message-driven beans that are acting as listeners on specific endpoints, which are destinations, and in managing the enterprise information system (EIS) resources that are utilized by these message-driven beans. Message-driven beans that are deployed as message endpoints are not the same as message-driven beans that are configured against a listener port. Message-driven beans that are used as message endpoints must be deployed using an ActivationSpecification that is defined within a resource adapter configuration for JCA Version 1.5.

With message endpoint MBeans, you can activate and deactivate specific endpoints within your applications to ensure that messages are delivered only to listening message-driven beans that are interacting with healthy EIS resources. This capability allows you to optimize the performance of your JMS applications in situations where an EIS resource is not behaving as expected. Message delivery to an endpoint typically fails when the message driven bean that is listening invokes an operation against a resource that is not healthy. For example, a messaging provider, which is an inbound resource adapter that is JCA Version 1.5 compliant, might fail to deliver messages to an endpoint when its underlying message-driven bean attempts to commit transactions against a database server that is not responding.

Note: Design your message-driven beans to delegate business processing to other enterprise beans. Do not access the EIS resources directly in the message-driven bean, but do so indirectly through a delegate bean.

Message endpoint MBeans alleviate two problems that are inherent to applications that provide message endpoints that access resources:

- Failed messages require additional processing, such as delivering them to the listening endpoint again or redirecting them to alternate destinations that process failed messages. In addition, a resource adapter might redeliver a message to an endpoint an infinite number of times.
- Message redirection requires the implementation of specialized destinations (queues and listeners) to process failed messages, as well as the logic to detect message failures. Message redirection is potentially error prone and computationally expensive due to its complexity.

The capability to deactivate (pause) and reactivate (resume) a specific message endpoint alleviates these problems by enabling the administrator to deactivate the endpoint from processing messages that are destined to fail. When the message endpoint is deactivated, you can repair the resource that is causing the problems and reactivate the endpoint to resume handling message requests. In the course of troubleshooting, you will not affect the resource adapter or the application that is hosting the endpoint.

1. Using the administrative console, navigate to the Message Endpoints panel for the application that is hosting the message endpoint.
   a. Select the **Applications** → **Application Types** → **Websphere enterprise applications** → ***application_name***.
   b. Select the **Runtime** panel.
   c. Select **Message Endpoints**. The panel lists the set of message endpoints that are hosted by the application.
2. Optional: Temporarily disable a message endpoint from handling messages and troubleshoot the problem.
   a. Deactivate the message endpoint by selecting the appropriate endpoint and clicking **Pause**.
   b. When the message endpoint is inactive, diagnose and repair the underlying cause of the delivery failures.
   c. Reactivate the message endpoint by selecting the appropriate endpoint and clicking **Resume**.

Results

The behavior you will observe when you deactivate (pause) a message endpoint using the message endpoint MBean is dependent upon a variety of factors, including the resource adapter that manages the message endpoint, the configuration of the message endpoint and the application server topology. Some specific examples of interest are as follows:

- **MDB listening on a non-durable topic (dependent on configuration):** The behavior that is implied by the deactivation (pause) of a message endpoint is often dependent upon the function that it is fulfilling. For example, if you have configured a message-driven bean to listen on a non-durable topic on the service integration bus, deactivating the message endpoint is analogous to stopping the application and will cause the subscription to be closed. This means that any messages that are published during the time that the message endpoint is paused will not be received by the message-driven bean.
- **Clustered message-driven bean (dependent on topology):** In this scenario a message-driven bean application has been deployed to a cluster of servers. A given message endpoint MBean controls only the behavior of the MDB in one server from the cluster, so will cause only one server to stop processing messages. Depending upon the messaging configuration and the specific resource adapter in use the messages that would have been consumed by the paused message endpoint may be consumed by the active message endpoints in the cluster, or they may remain unconsumed until the paused message endpoint is resumed.
- **Clustered message-driven bean, a non-clustered queue:** In this scenario, you have a cluster of servers with the same message-driven bean deployed to them. This is similar to the case, in which you have different message-driven beans with the same message selection criteria, except that in this case the message-driven beans are logically the same message-driven bean. Pausing the endpoint will cause only one of the servers to stop receiving messages, and the other message-driven beans will

receive all the messages; none of the messages will be orphaned. To stop all of the endpoints, you must direct each server in the cluster to stop the local message endpoint.

- **Clustered message-driven bean, clustered queue:** In this scenario, each message-driven bean is pulling messages from a different partition of the queue. Messaging through WebSphere MQ and the Service Integration Bus have similar, but different, capabilities. If you are using WebSphere MQ, then pausing one endpoint will not allow the other instances of the message-driven bean to receive the messages. In the Service Integration Bus, messages from a paused endpoint will be redirected to the other message-driven beans.

Related tasks

"Managing the message endpoint lifecycle using scripting" on page 1361
Use the Jython scripting language to manage your message endpoints with the wsadmin tool. Use this topic to query your configuration for message endpoint properties, and to deactivate or reactivate a message endpoint.

Manage message endpoints

Use this panel to manage situations where messaging providers fail to deliver messages to their intended destinations. For example, a provider might fail to deliver messages to a message endpoint when its underlying message driven bean attempts to commit transactions against a database server that is not responding.

To view this administrative console panel:

1. Select the **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name***.
2. Select the **Runtime** panel. You will only see the Runtime panel if you have an application installed that is hosting message-driven beans.
3. Select **Message Endpoints**. The panel lists the set of message endpoints that are hosted by the application.

Name:

Specifies the name of the message endpoint.

Click the name of the message endpoint to view the configuration binding for the underlying endpoint message-driven bean and Activation Specification.

Running object scope:

Specifies the server where the endpoint is running.

For more information on scope, see Administrative console scope settings.

Status:

Indicates whether the message endpoint is active or inactive.

Click **Pause** to deactivate a message endpoint and stop it from handling messages.

Click **Resume** to reactivate a message endpoint that is inactive.

Configuring a JDBC provider and data source

For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

Before you begin

Each vendor database requires different JDBC driver implementation classes for JDBC connectivity. A JDBC provider encapsulates those vendor-specific driver files. Through the data source that you associate with the JDBC provider, an application server obtains and manages the physical connections for transactions between applications and the database.

Note: If you are accessing a DB2 database, Data Studio pureQuery is an alternative to JDBC. For more information on pureQuery, see the topic ″Task overview: Data Studio pureQuery″ in the related links section.

Determine the version of data source that you need according to the API specification of your applications.

- *Data sources (WebSphere Application Server Version 4)* are for use with the Enterprise JavaBeans (EJB) 1.0 specification and the Java Servlet 2.2 specification.
- Data sources of the latest standard version are for use with applications that implement the more advanced releases of these specifications.

1. Verify that all of the necessary JDBC driver files are installed on your application server. Consult the article “Data source minimum required settings, by vendor” on page 1433 for that information. If you opt to configure a user-defined JDBC provider, check your database documentation for information about the driver files.
2. Create a JDBC provider.

   From the administrative console, see Creating a JDBC provider using the administrative console.

   OR

   Using the wsadmin scripting client, see “Configuring a JDBC provider using scripting” on page 1338.

   OR

   Using the Java Management Extensions (JMX) API, see Creating a JDBC provider and data source using the Java Management Extensions API.
3. Create a data source.

   From the administrative console, see Creating a data source using the administrative console.

   OR

   Using the wsadmin scripting client, see “Configuring new data sources using scripting” on page 1340. (For V4 data sources, see “Configuring new WAS40 data sources using scripting” on page 1350.)

   OR

   Using the JMX API, see Creating a JDBC provider and data source using the Java Management Extensions API.

   Note: Different database vendors require different properties for implementations of their JDBC drivers. Set these properties on the WebSphere Application Server data source. Because Application Server contains templates for many vendor JDBC implementations, the administrative console surfaces the required properties and prompts you for them as you create a data source. However, if you script your data access configurations, you must consult the article “Data source minimum required settings, by vendor” on page 1433 for the required properties and settings options.
4. Optional: Configure custom properties. Like the required properties, custom properties for specific vendor JDBC drivers must be set on the Application Server data source. Consult your database documentation for information about available custom properties. To configure a custom class to facilitate the handling of database properties that are not recognized natively by the Application Server, refer to “Developing a custom DataStoreHelper class” on page 1519

   You can also learn about optional data source properties in the Application Programming Guide and Reference for Java for your version of DB2 for z/OS if you use the DB2 Universal JDBC Driver provider.

5. Bind resource references to the data source. See the article Data source lookups for enterprise beans and Web modules .
6. Test the connection (for non-container-managed persistence usage). See the “Test connection service” on page 1521 article.

Results

If you use the DB2 JDBC Universal Driver, you might experience data source failures that the application server JVM log does not document. Check the DB2 database log or the WebSphere Application Server JDBC trace log (if JDBC trace was active). You might find that a bad authentication credential is the cause of failure. Currently the DB2 JDBC Universal Driver does not identify or surface the errors that are produced by non-valid authentication credentials in a proper or consistent way.

Even if you receive information about a bad credential, check the database and JDBC trace logs. These logs provide more reliable, detailed error data on authentication failures.

Note: The JDBC trace log exists only if the JDBC trace service is active during server start up. Activate the service in the administrative console. For more information, see Enabling trace at server startup. Specify **WAS.database** as the trace group and select **com.ibm.ws.db2.logwriter** as the trace string.

Data source minimum required settings, by vendor

These properties vary according to the database vendor requirements for Java Database Connectivity (JDBC) driver implementations. You must set the appropriate properties on every data source that you configure.

Use these tables for quick reference on the JDBC providers that represent your JDBC driver classes. Each table corresponds to a specific database vendor, product, and platform.

Following the tables are links to detailed requirements for creating data sources that correspond to each JDBC provider that the application server supports . The list includes information about connection properties that are required by the database and any optional properties that the JDBC driver supports. Use the administrative console or the wsadmin scripting tool to define these properties on your data sources.

Apache Derby		
JDBC provider	**Transaction support**	**Version and other considerations**
Derby JDBC Provider	One-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.0.2 and later • Not for use in clustered environment: accessible from a single JVM only
Derby JDBC Provider (XA)	One and two-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.0.2 and later • Not for use in clustered environment: accessible from a single JVM only

Apache Derby		
JDBC provider	**Transaction support**	**Version and other considerations**
Derby JDBC Provider 40	One-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources
Derby JDBC Provider 40 (XA)	One and two-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources
Derby Network Server using Derby Client	One-phase	• Does not support Version 4.0 data sources. • Configurable only in nodes at version 6.1 and later • *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs • Only for use with Apache Derby databases that run on the same node as the application server
Derby Network Server using Derby Client (XA)	One and two-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.1 and later • *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs • Only for use with Apache Derby databases that run on the same node as the application server
Derby Network Server using Derby Client 40	One-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources
Derby Network Server using Derby Client 40 (XA)	One and two-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources

DB2 on AIX, HP-UX, Linux, Solaris, and Windows systems		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 Using IBM JCC Driver	One-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Using IBM JCC Driver (XA)	One and two-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Universal JDBC Provider	One-phase	N/A
DB2 Universal JDBC Provider (XA)	One and two-phase	N/A

DB2 UDB for iSeries		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 UDB for iSeries (Native)	One-phase	Recommended when you run the application server on iSeries.
DB2 UDB for iSeries (Native XA)	One and two-phase	Recommended when you run the application server on iSeries.
DB2 UDB for iSeries (Toolbox)	One-phase	N/A
DB2 UDB for iSeries (Toolbox XA)	One and two-phase	N/A

DB2 on z/OS		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 Using IBM JCC Driver	One-phase	• Configurable in version 7.0 and later nodes.
DB2 Using IBM JCC Driver (XA)	One and two-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Universal JDBC Provider	One-phase when connecting to the application server that is on AIX, HP-UX, Linux, Solaris, and Windows systems	
DB2 Universal JDBC Provider (XA)	One and two-phase	

Informix		
JDBC provider	**Transaction support**	**Version and other considerations**
Informix Using IBM JCC Driver	One phase	This provider is configurable in nodes that are at version 7.0 and later.
Informix Using IBM JCC Driver (XA)	One and two-phase	This provider is configurable in nodes that are at version 7.0 and later.
Informix JDBC Provider	One-phase	N/A
Informix JDBC Provider (XA)	One and two-phase	N/A

Microsoft SQL Server		
JDBC provider	**Transaction support**	**Version and other considerations**
Microsoft SQL Server JDBC Driver	One-phase	N/A
Microsoft SQL Server JDBC Driver (XA)	One and two-phase	N/A
DataDirect ConnectJDBC Provider, type 4 driver, for Microsoft SQL Server	One-phase	N/A
DataDirect ConnectJDBC Provider, type 4 driver, for Microsoft SQL Server (XA)	One and two-phase	N/A

Oracle		
JDBC provider	**Transaction support**	**Version and other considerations**
Oracle JDBC Provider	One-phase	N/A
Oracle JDBC Provider (XA)	One and two-phase	N/A

Sybase		
JDBC provider	**Transaction support**	**Version and other considerations**
Sybase JDBC 3 Driver	One-phase	jConnect v6.05
Sybase JDBC 3 Driver (XA)	One and two-phase	jConnect v6.05
Sybase JDBC 2 Driver	One-phase	jConnect v5.5
Sybase JDBC 2 Driver (XA)	One and two-phase	jConnect v5.5

Detailed requirements

The following list identifies required class files and connection properties per JDBC provider.

After you determine the JDBC provider that suits your application and environment, ensure that you acquire the corresponding JDBC driver at a release level supported by this version of the application server. Consult the IBM support website for supported hardware and software.

Use the following links to navigate to the requirements list. Each link corresponds to a specific database vendor, product, and platform.

- Apache Derby or Cloudscape 10.x
- DB2 Universal Database for iSeries
- Informix
- Microsoft SQL Server
- Oracle
- Sybase

Data source minimum required settings for Apache Derby:

These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Apache Derby and Cloudscape data sources.

You can configure the following types of providers:

- Derby JDBC Provider
- Derby JDBC Provider (XA)
- Derby JDBC Provider 40
- Derby JDBC Provider 40 (XA)
- Derby Network Server using Derby Client
- Derby Network Server using Derby Client (XA)
- Derby Network Server using Derby Client 40
- Derby Network Server using Derby Client 40 (XA)
- **Derby JDBC Provider**

  The Derby JDBC driver provides JDBC access to the Apache Derby database by using the framework that is already embedded in the application server. You cannot use any Version 4.0 data sources with this provider.

  This provider:

  – Is configurable only in nodes at version 6.0.2 and later

  – Supports one phase data source with the following class:

  ```
  org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource
  ```

– Requires the JDBC driver file:
  - derby.jar

    The full path name is ${DERBY_JDBC_DRIVER_PATH}/derby.jar. When you create a connection through the application server, the environment variables are set automatically.
– Requires the following DataStoreHelper class:

  `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`
– Requires the following properties:

  databaseName

  The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

  If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

  `c:\temp\sampleDB;create=true`

- **Derby JDBC Provider (XA)**

  The Derby JDBC driver (XA) provides JDBC access to the Apache Derby database by using the framework that is already embedded in the application server.

  This provider:
  – Does not support use Version 4.0 data sources.
  – Is configurable only in nodes at version 6.0.2 and later
  – Supports the two-phase data source with the following class:

    `org.apache.derby.jdbc.EmbeddedXADataSource`
  – Requires JDBC driver file:
    - derby.jar

      The full path name is ${DERBY_JDBC_DRIVER_PATH}/derby.jar. When you create a connection through the application server, the environment variables are set automatically.
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`
  – Does not require a valid authentication alias.
  – Requires the following properties:

    databaseName

    The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

    If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

    `c:\temp\sampleDB;create=true`

- **Derby JDBC Provider 40**

  The Derby JDBC Provider 40 provides JDBC access to the Apache Derby database by using the framework that is already embedded in the application server.

  This provider:
  – Is configurable only in nodes at version 7.0 and later.
  – Does not support Version 4.0 data sources.
  – Supports one phase data source with the following class:

    `org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource40`

– Requires the JDBC driver file:
  - derby.jar

    The full path name is ${DERBY_JDBC_DRIVER_PATH}/derby.jar. When you create a connection through the application server, the environment variables are set automatically.
– Requires the following DataStoreHelper class:

  `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`
– Requires the following properties:

  databaseName

  The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

  If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

  `c:\temp\sampleDB;create=true`

- **Derby JDBC Provider 40 (XA)**

  The Derby JDBC Provider 40 (XA) provides JDBC access to the Apache Derby database by using the framework that is already embedded in the application server.

  This provider:
  – Is configurable only in nodes at version 7.0 and later.
  – Does not support Version 4.0 data sources.
  – Supports one phase data source with the following class:

    `org.apache.derby.jdbc.EmbeddedXADataSource40`
  – Requires the JDBC driver file:
    - derby.jar

      The full path name is ${DERBY_JDBC_DRIVER_PATH}/derby.jar. When you create a connection through the application server, the environment variables are set automatically.
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DerbyDataStoreHelper`
  – Requires the following properties:

    databaseName

    The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

    If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

    `c:\temp\sampleDB;create=true`

- **Derby Network Server using Derby Client**

  Use this provider to access only Apache Derby databases that run on the same node as the application server.

  This provider:
  – Does not support Version 4.0 data sources.
  – Is configurable only in nodes at version 6.1 and later
  – Uses the following one phase data source for the Derby Network Server using Derby Client provider:

    `org.apache.derby.jdbc.ClientConnectionPoolDataSource`
  – Requires the following JDBC driver file:

- derbyclient.jar

– Requires DataStoreHelper class:

`com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

– Requires the following property:

databaseName

The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

`c:\temp\sampleDB;create=true`

- **Derby Network Server using Derby Client (XA)**

Use this provider to access only Apache Derby databases that run on the same node as the application server.

This provider:

– Does not support Version 4.0 data sources.

– Is configurable only in nodes at version 6.1 and later

– Uses the following XA data source for this Derby Network Server using Derby Client provider:

`org.apache.derby.jdbc.ClientXADataSource`

– Requires the following JDBC driver file:

- derbyclient.jar

– Requires the DataStoreHelper class:

`com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

– Requires the following property:

databaseName

The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

`c:\temp\sampleDB;create=true`

- **Derby Network Server using Derby Client 40**

Use this provider to access only Apache Derby databases that run on the same node as the application server.

This provider:

– Is configurable only in nodes at version 7.0 and later

– Does not support Version 4.0 data sources.

– Uses the following one phase data source for the Derby Network Server using Derby Client provider:

`org.apache.derby.jdbc.ClientConnectionPoolDataSource40`

– Requires the following JDBC driver file:

- derbyclient.jar

– Requires DataStoreHelper class:

`com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

– Requires the following property:

databaseName

The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

`c:\temp\sampleDB;create=true`

- **Derby Network Server using Derby Client 40 (XA)**

  Use this provider to access only Apache Derby databases that run on the same node as the application server.

  This provider:

  – Is configurable only in nodes at version 7.0 and later

  – Does not support Version 4.0 data sources.

  – Uses the following one phase data source for the Derby Network Server using Derby Client provider:

  `org.apache.derby.jdbc.ClientXADataSource40`

  – Requires the following JDBC driver file:

  - derbyclient.jar

  – Requires DataStoreHelper class:

  `com.ibm.websphere.rsadapter.DerbyNetworkServerDataStoreHelper`

  – Requires the following property:

  databaseName

  The name of the database from which the data source obtains connections. If you do not specify a fully qualified path name, the application server uses the default location of *app_server_root*/derby or the equivalent default for AIX, HP-UX, Linux, or Solaris system environments. An example for the database path name is:

  If no database exists for the path name that you want to specify, append `;create=true` to the path name to create a database dynamically. For example:

  `c:\temp\sampleDB;create=true`

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

“Configuring a JDBC provider using the administrative console” on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

“Configuring a data source using the administrative console” on page 1459
Application components use a data source to access connection instances to a relational database.

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Related information

http://publib.boulder.ibm.com/infocenter/cscv/v10r1/index.jsp

Data source minimum required settings for DB2 Universal Database for iSeries:

These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for a DB2 UDB data source.

What type of configuration do you have?

- "DB2 UDB for iSeries with the application server for AIX, HP-UX, i5/OS, Linux, Solaris, or Windows"

DB2 UDB for iSeries with the application server for AIX, HP-UX, i5/OS, Linux, Solaris, or Windows

You can configure the following types of providers:

- DB2 UDB for iSeries (Native)
- DB2 UDB for iSeries (Native XA)
- DB2 UDB for iSeries (Toolbox)
- DB2 UDB for iSeries (Toolbox XA)
- **DB2 UDB for iSeries (Native)**

  The iSeries Developer Kit for Java contains this Type 2 JDBC driver that is built on top of the iSeries DB2 Call Level Interface (CLI) native libraries.

  This provider:

  – Is for local DB2 connections on iSeries. It is not recommended for remote access.
  – Supports the one-phase data source:

    `com.ibm.db2.jdbc.app.UDBConnectionPoolDataSource`
  – Requires the following JDBC driver files:
    - db2_classes16.jar - for nodes that are running at Version 7.0 or later. The location of the jar file is /QIBM/Proddata/java400/jdk6/lib/ext/db2_classes16.jar.
    - db2_classes.jar - for nodes that are running at Version 6.1 or earlier. The location of the jar file is /QIBM/ProdData/Java400/ext/db2_classes.jar.
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`
  – Does not require an authentication alias.
  – Requires the following properties:
    - **databaseName** - The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.
- **DB2 UDB for iSeries (Native XA)**

  The iSeries Developer Kit for Java contains this XA-compliant Type 2 JDBC driver built on top of the iSeries DB2 Call Level Interface (CLI) native libraries.

  This provider:

  – Is for local DB2 connections on iSeries. It is not recommended for remote access.
  – Supports the following two-phase data source:

    `com.ibm.db2.jdbc.app.UDBXADataSource`
  – Requires the following JDBC driver files:
    - db2_classes16.jar - for nodes that are running at Version 7.0 or later. The location of the jar file is /QIBM/Proddata/java400/jdk6/lib/ext/db2_classes16.jar.
    - db2_classes.jar - for nodes that are running at Version 6.1 or earlier. The location of the jar file is /QIBM/ProdData/Java400/ext/db2_classes.jar.
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`
  – Does not require an authentication alias.

– Requires the following properties:
  - **databaseName** - The name of the relational database to which the data source connections are established. This name must appear in the iSeries Relational Database Directory. The default is *LOCAL.

- **DB2 UDB for iSeries (Toolbox)**

  This JDBC driver, also known as iSeries Toolbox driver for Java, is provided in the DB2 for iSeries database server.

  This provider:

  – Is for remote DB2 connections on iSeries. Use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.
  – Supports the following one-phase data source:

    `com.ibm.as400.access.AS400JDBCConnectionPoolDataSource`
  – Requires the following JDBC driver files:
    - jt400.jar
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`
  – Does not require an authentication alias if the application server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.
  – Requires the following properties:
    - **serverName** - The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

- **DB2 UDB for iSeries (Toolbox XA)**

  This XA compliant JDBC driver, also known as iSeries Toolbox XA compliant driver for Java, is provided in the DB2 for iSeries database server.

  This provider:

  – Is for remote DB2 connections on iSeries. Use this driver instead of the IBM Developer Kit for Java JDBC Driver to access remote DB2 UDB for iSeries systems.
  – Supports the following two-phase data source:

    `com.ibm.as400.access.AS400JDBCXADataSource`
  – Requires the following JDBC driver files:
    - jt400.jar
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper`
  – Does not require an authentication alias if the application server and DB2 UDB for iSeries are installed in the same server. If they are installed in different servers, the user ID and password are required.
  – Requires the following properties:
    - **serverName** - The name of the server from which the data source obtains connections. Example: *myserver.mydomain.com*.

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

“Configuring a JDBC provider using the administrative console” on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

“Configuring a data source using the administrative console” on page 1459
Application components use a data source to access connection instances to a relational database.

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Related information

DB2 Universal Database for iSeries support pages

DB2 for iSeries product Web pages

Data source minimum required settings for Informix:

These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Informix data sources.

You can configure the following types of providers:

- Informix Using IBM JCC Driver
- Informix Using IBM JCC Driver (XA)
- Informix Using IBM Universal JDBC Driver
- Informix Using IBM Universal JDBC Driver (XA)
- **Informix Using IBM JCC Driver**

  The Informix Using IBM JCC Driver is a one-phase commit provider for Informix that uses the IBM Data Server Driver for JDBC and SQLJ. The IBM Data Server Driver is JDBC 4.0 compliant and is the next generation of the Universal JCC driver.

  This provider:

  – Is configurable in version 7.0 and later nodes.
  – Supports the following one-phase data source:

    ```
    com.ibm.db2.jcc.DB2ConnectionPoolDataSource
    ```

  – Requires the following JDBC driver files and class path settings:

    ```
    ${INFORMIX_JCC_DRIVER_PATH}/db2jcc4.jar
    ${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
    ```

  – Requires the following DataStoreHelper class:

    ```
    com.ibm.websphere.rsadapter.InformixJccDataStoreHelper
    ```

  – Requires a valid authentication alias.
  – Requires the following properties:
    - **serverName** - The TCP/IP address or host name for the DRDA server. The name of the Informix instance on the server.
    - **portNumber** - The TCP/IP port number where the DRDA server resides.

- **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.

- **Informix Using IBM JCC Driver (XA)**

  The Informix Using IBM JCC Driver (XA) is a two-phase commit provider for Informix that uses the IBM Data Server Driver for JDBC and SQLJ. The IBM Data Server Driver is JDBC 4.0 compliant and is the next generation of the Universal JCC driver.

  This provider:

  – Is configurable in version 7.0 and later nodes.

  – Supports two phase data source:

    ```
    com.ibm.db2.jcc.DB2XADataSource
    ```

  – Requires JDBC driver files:

    ```
    db2jcc4.jar
    db2jcc_license_cu.jar
    db2jcc_license_cisuz.jar
    ```

    Note: If you plan to use SQLJ for queries, this provider also requires driver file **ifxsqlj.jar**.

  – Requires the following DataStoreHelper class:

    ```
    com.ibm.websphere.rsadapter.InformixJccDataStoreHelper
    ```

  – Requires a valid authentication alias.

  – Requires the following properties:

    - **serverName** - The TCP/IP address or host name for the DRDA server. The name of the Informix instance on the server.
    - **portNumber** - The TCP/IP port number where the DRDA server resides.
    - **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.

  Note: You cannot use Informix XA data sources with ANSI databases if SQL statements are issued in local transactions instead of global transactions. This scenario might occur within the application code or within a component of Application Server such as scheduler. The following message might be logged if you are experiencing this problem:

  ```
  java.sql.SQLException: Already in transaction.
         at com.informix.util.IfxErrMsg.getSQLException(IfxErrMsg.java:398)
         at com.informix.jdbc.IfxSqli.a(IfxSqli.java:3247)
         at com.informix.jdbc.IfxSqli.E(IfxSqli.java:3556)
         at com.informix.jdbc.IfxSqli.dispatchMsg(IfxSqli.java:2382)
         at com.informix.jdbcx.IfxXASqli.receiveMessage(IfxXASqli.java:120)
         at com.informix.jdbc.IfxSqli.X(IfxSqli.java:7926)
         at com.informix.jdbc.IfxSqli.a(IfxSqli.java:854)
         at com.informix.jdbc.IfxSqli.executeCommand(IfxSqli.java:749)
         at com.informix.jdbc.IfxResultSet.b(IfxResultSet.java:293)
         at com.informix.jdbc.IfxStatement.c(IfxStatement.java:1269)
         at com.informix.jdbc.IfxStatement.b(IfxStatement.java:423)
         at com.informix.jdbc.IfxStatement.executeUpdate(IfxStatement.java:277)
         at com.informix.jdbc.IfxSqliConnect.setTransactionIsolation(IfxSqliConnect.java:2565)
  ```

  To avoid this issue:

  – Switch to a non-ANSI database.

  – If the error is triggered by an application, update the application such that it always runs in a global transaction.

- **Informix Using IBM Universal JDBC Driver**

  The Informix JDBC Driver is a Type 4 JDBC driver that is JDBC 3.0 compliant and provides access to the Informix database.

  This provider:

– Supports the following one-phase data source:

```
com.ibm.db2.jcc.DB2ConnectionPoolDataSource
```

– Requires the following JDBC driver files:

```
db2jcc.jar
db2jcc_license_cu.jar
db2jcc_license_cisuz.jar
```

– Requires the following DataStoreHelper class:

```
com.ibm.websphere.rsadapter.InformixJccDataStoreHelper
```

– Requires a valid authentication alias.

– Requires the following properties:

- **serverName** - The TCP/IP address or host name for the DRDA server. The name of the Informix instance on the server.
- **portNumber** - The TCP/IP port number where the DRDA server resides.
- **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.

• **Informix Using IBM Universal JDBC Driver (XA)**

The Informix JDBC Driver (XA) is a Type 4 JDBC driver that is JDBC 3.0 compliant and provides XA-compliant JDBC access to the Informix database.

This provider:

– Supports the following two-phase data source:

```
com.ibm.db2.jcc.DB2XADataSource
```

– Requires the following JDBC driver files:

```
db2jcc.jar
db2jcc_license_cu.jar
db2jcc_license_cisuz.jar
```

– Requires **DataStoreHelper** class:

```
com.ibm.websphere.rsadapter.InformixJccDataStoreHelper
```

– Requires a valid authentication alias.

– Requires the following properties:

- **serverName** - The TCP/IP address or host name for the DRDA server. The name of the Informix instance on the server.
- **portNumber** - The TCP/IP port number where the DRDA server resides.
- **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.

Note: You cannot use Informix XA data sources with ANSI databases if SQL statements are issued in local transactions instead of global transactions. This scenario might occur within the application code or within a component of Application Server such as scheduler. The following message might be logged if you are experiencing this problem:

```
java.sql.SQLException: Already in transaction.
       at com.informix.util.IfxErrMsg.getSQLException(IfxErrMsg.java:398)
       at com.informix.jdbc.IfxSqli.a(IfxSqli.java:3247)
       at com.informix.jdbc.IfxSqli.E(IfxSqli.java:3556)
       at com.informix.jdbc.IfxSqli.dispatchMsg(IfxSqli.java:2382)
       at com.informix.jdbcx.IfxXASqli.receiveMessage(IfxXASqli.java:120)
       at com.informix.jdbc.IfxSqli.X(IfxSqli.java:7926)
       at com.informix.jdbc.IfxSqli.a(IfxSqli.java:854)
       at com.informix.jdbc.IfxSqli.executeCommand(IfxSqli.java:749)
       at com.informix.jdbc.IfxResultSet.b(IfxResultSet.java:293)
       at com.informix.jdbc.IfxStatement.c(IfxStatement.java:1269)
       at com.informix.jdbc.IfxStatement.b(IfxStatement.java:423)
       at com.informix.jdbc.IfxStatement.executeUpdate(IfxStatement.java:277)
       at com.informix.jdbc.IfxSqliConnect.setTransactionIsolation(IfxSqliConnect.java:2565)
```

To avoid this issue:

– Switch to a non-ANSI database.

– If the error is triggered by an application, update the application such that it always runs in a global transaction.

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

“Configuring a JDBC provider using the administrative console” on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

“Configuring a data source using the administrative console” on page 1459
Application components use a data source to access connection instances to a relational database.

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Related information

IBM Informix product family Web pages

Data source minimum required settings for Microsoft SQL Server:

These properties vary according to the database vendor requirements for Java Database Connectivity (JDBC) driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Microsoft SQL Server data sources.

Note: If you use only the features of Microsoft SQL Server that have no impact on JDBC transactions, you generally risk no exceptions by upgrading to a newer version.

The application server supports the SNAPSHOT and READ_COMMITTED_SNAPSHOT isolation levels in Microsoft SQL Server. This chart describes the SNAPSHOT and READ_COMMITTED_SNAPSHOT isolation levels for different types of providers:

JDBC provider	MS SQL Server feature	Configuration consideration
Microsoft SQL Server JDBC Driver	SNAPSHOT isolation level	Set the isolation level constant by invoking the setTransactionIsolation method: • conn.setTransactionIsolation (com.microsoft.sqlserver.jdbc.SQLServerConnection.TRANSACTION_ SNAPSHOT) • conn.setTransactionIsolation(*value_of_constant*)
	READ_COMMITTED_SNAPSHOT isolation level	This isolation level is an implementation of the Read committed isolation level. The policy enforces optimistic locking for read operations with MS SQL Server 2005. 1. Configure the isolation level on the database 2. Invoke the setTransactionIsolation method with the conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_READ_COMMITTED) attribute.

JDBC provider	MS SQL Server feature	Configuration consideration
DataDirect ConnectJDBC type 4 driver for MS SQL Server	SNAPSHOT isolation level	This isolation level implements optimistic locking for transactions in which MS SQL Server 2005 serializes the data. Configure the ALLOW_SNAPSHOT_ISOLATION setting on the database, and then set the isolation level in one of two ways: • By isolation level constant. Invoke the setTransactionIsolation method with one of two attributes: – conn.setTransactionIsolation (com.ddtek.jdbc.extensions. ExtConstants.TRANSACTION_SNAPSHOT) – conn.setTransactionIsolation(16) • By the custom data source property: – Set the data source custom property snapshotSerializable to `true`.
	READ_COMMITTED_SNAPSHOT isolation level	This isolation level is an implementation of the Read committed isolation level. The policy enforces optimistic locking for read operations with MS SQL Server 2005. 1. Configure the isolation level on the database 2. Invoke the setTransactionIsolation method with the conn.setTransactionIsolation (java.sql.Connection.TRANSACTION_READ_COMMITTED) attribute.

You can configure the following types of providers:

- Microsoft SQL Server JDBC Driver
- Microsoft SQL Server JDBC Driver (XA)
- DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server
- DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server (XA)
- **Microsoft SQL Server JDBC Driver**

  This provider:
  – Supports the following data source:

    `com.microsoft.sqlserver.jdbc.SQLServerConnectionPoolDataSource`
  – Requires the following Java archive (JAR) files:
    - sqljdbc.jar
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.MicrosoftSQLServerDataStoreHelper`
  – Requires a valid authentication alias.
  – Requires the following properties:
    - **serverName** - specifies the name of the server in which Microsoft SQL Server resides. Example: *myserver.mydomain.com*
    - **portNumber** - specifies the TCP/IP port that Microsoft SQL Server uses for communication. Port 1433 is the default.
    - **databaseName** - specifies the name of the database from which the data source obtains connections. Example: *Sample*.
- **Microsoft SQL Server JDBC Driver (XA)**

  This provider:
  – Supports the following data source:

    `com.microsoft.sqlserver.jdbc.SQLServerXADataSource`
  – Requires the following Java archive (JAR) files:
    - sqljdbc.jar
  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.MicrosoftSQLServerDataStoreHelper`
  – Requires the following properties:
    - **serverName** - specifies the name of the server in which Microsoft SQL Server resides. Example: *myserver.mydomain.com*
    - **portNumber** - specifies the TCP/IP port that Microsoft SQL Server uses for communication. Port 1433 is the default.

- **databaseName** - specifies the name of the database from which the data source obtains connections. Example: *Sample*.

- **DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server**

  DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server is a Type 4 JDBC driver that provides JDBC access to the Microsoft SQL Server database. Use this provider only with a Connect JDBC driver that is purchased from DataDirect Technologies.

  The JDBC provider:

  – Supports the following data source:

    `com.ddtek.jdbcx.sqlserver.SQLServerDataSource`

  – Requires the following JDBC driver files:

    - sqlserver.jar

  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

  – Requires a valid authentication alias.

  – Requires the following properties:

    - **serverName** - specifies the name of the server in which Microsoft SQL Server resides. Example: *myserver.mydomain.com*
    - **portNumber** - specifies the TCP/IP port that Microsoft SQL Server uses for communication. Port 1433 is the default.
    - **databaseName** - specifies the name of the database from which the data source obtains connections. Example: *Sample*.

- **DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server (XA)**

  DataDirect ConnectJDBC type 4 driver for Microsoft SQL Server (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Microsoft SQL Server database. Use this provider only with a Connect JDBC driver that is purchased from DataDirect Technologies.

  This provider:

  – Supports the following data source:

    `com.ddtek.jdbcx.sqlserver.SQLServerDataSource`

  – Requires the following JDBC driver files:

    - sqlserver.jar

  – Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.ConnectJDBCDataStoreHelper`

  – Requires a valid authentication alias.

  – Requires the following properties:

    - **serverName** - specifies the name of the server in which Microsoft SQL Server resides. Example: *myserver.mydomain.com*
    - **portNumber** - specifies the TCP/IP port that Microsoft SQL Server uses for communication. Port 1433 is the default.
    - **databaseName** - specifies the name of the database from which the data source obtains connections. Example: *Sample*.

Related tasks

"Configuring a JDBC provider and data source" on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

"Configuring a JDBC provider using the administrative console" on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

"Configuring a data source using the administrative console" on page 1459
Application components use a data source to access connection instances to a relational database.

"Creating and configuring a JDBC provider and data source using the Java Management Extensions API" on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Related reference

Microsoft SQL Server 2005 JDBC Driver

Data source minimum required settings for Oracle:

These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Oracle data sources.

You can configure the following types of providers:

- Oracle JDBC Driver
- Oracle JDBC Driver (XA)
- **Oracle JDBC Driver**

  The Oracle JDBC Driver provides JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

  This provider:
  - Supports one-phase data source:

    `oracle.jdbc.pool.OracleConnectionPoolDataSource`
  - Requires the following JDBC driver files:
    - ojdbc6.jar.

      Note: Be aware of the following:
      - Oracle does not support the use of ojdbc14.jar in an environment with the Java SE Development Kit Version 6 or later.
      - In mixed node environments, the data source wizard in the administrative console allows you to choose a class path for ojdbc6.jar or ojdbc14.jar.
      - For Oracle trace, use ojdbc*version*_g.jar.
  - Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper`

    Note: You must use the Oracle11gDataStoreHelper with the ojdbc6.jar driver file, regardless of whether you use an Oracle 11g or Oracle 10g database server.
  - Requires a valid authentication alias.
  - Requires properties:

    URL The URL that indicates the database from which the data source obtains connections. For example:

    `jdbc:oracle:thin:@myServer:1521:myDatabase`

where *myServer* is the server name, *1521* is the port that the server uses for communication, and *myDatabase* is the database name.

- **Oracle JDBC Driver (XA)**

  The Oracle JDBC Driver (XA) provides XA-compliant JDBC access to the Oracle database. This JDBC driver supports both Type 2 JDBC access and Type 4 JDBC access.

  This provider:
  - Supports two-phase data source:

    `oracle.jdbc.xa.client.OracleXADataSource`
  - Requires the following JDBC driver files:
    - ojdbc6.jar.

      Note: Be aware of the following notes regarding the use of these driver files:
      - Oracle does not support the use of ojdbc14.jar in an environment with the Java SE Development Kit Version 6 or later.
      - In mixed node environments, the data source wizard in the administrative console allows you to choose a class path for ojdbc6.jar or ojdbc14.jar.
      - For Oracle trace, use ojdbc*version*_g.jar.
  - Requires the following DataStoreHelper class:

    `com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper`

    Note: You must use the Oracle11gDataStoreHelper with the ojdbc6.jar driver file, regardless of whether you use an Oracle 11g or Oracle 10g database server.
  - Requires a valid authentication alias.
  - Requires properties:

    URL Indicates the database from which the data source obtains connections. For example:

    `jdbc:oracle:thin:@myServer:1521:myDatabase`

    where *myServer* is the server name, *1521* is the port that the server uses for communication, and *myDatabase* is the database name.

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

“Configuring a JDBC provider using the administrative console” on page 1453
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

“Configuring a data source using the administrative console” on page 1459
Application components use a data source to access connection instances to a relational database.

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Related reference

Oracle Technology Network - SQLJ/JDBC Download Page

Data source minimum required settings for Sybase:

These properties vary according to the database vendor requirements for JDBC driver implementations. You must set the appropriate properties on every data source that you configure. These settings are for Sybase data sources.

You can configure the following types of providers:

- Sybase JDBC 3 Driver
- Sybase JDBC 3 Driver (XA)
- Sybase JDBC 2 Driver
- Sybase JDBC 2 Driver (XA)
- **Sybase JDBC 3 Driver**

  The Sybase JDBC 3 Driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

  This provider:

  – Uses jConnect Version 6.05

  – Supports one phase data source:

  `com.sybase.jdbc3.jdbc.SybConnectionPoolDataSource`

  – Requires the following JDBC driver files:

  `jconn3.jar`

  – Requires the following DataStoreHelper class:

  `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

  – Requires a valid authentication alias.

  – Requires the following properties:

  - **serverName** - The name of the database server. Example: *myserver.mydomain.com*.
  - **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.
  - **portNumber** - The TCP/IP port number through which all communications to the server take place. Example: *5000*.
  - **connectionProperties** - A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true` (Type: java.lang.String)

- **Sybase JDBC 3 Driver (XA)**

  The Sybase JDBC 3 Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

  This provider:

  – Uses jConnect Version 6.05

  – Supports two phase data source:

  `com.sybase.jdbc3.jdbc.SybXADataSource`

  – Requires JDBC driver files:

  `jconn3.jar`

  – Requires the following DataStoreHelper class:

  `com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

  – Requires a valid authentication alias.

  – Requires the following properties:

  - **serverName** - The name of the database server. Example: *myserver.mydomain.com*
  - **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.
  - **portNumber** - The TCP/IP port number through which all communications to the server take place. Example: *5000*.
  - **connectionProperties** - A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true` (Type: java.lang.String)

- **Sybase JDBC 2 Driver**

The Sybase JDBC 2 Driver is a Type 4 JDBC driver that provides JDBC access to the Sybase database.

This provider:

– Uses jConnect Version 5.5

– Supports one phase data source:

`com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource`

– Requires the following JDBC driver files:

`jconn2.jar`

– Requires the following DataStoreHelper class:

`com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

– Requires a valid authentication alias.

– Requires the following properties:

- **serverName** - The name of the database server. Example: *myserver.mydomain.com*.
- **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.
- **portNumber** - The TCP/IP port number through which all communications to the server take place. Example: *5000*.
- **connectionProperties** - A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true` (Type: java.lang.String)

• **Sybase JDBC 2 Driver (XA)**

The Sybase JDBC 2 Driver (XA) is a Type 4 JDBC driver that provides XA-compliant JDBC access to the Sybase database.

This provider:

– Uses jConnect Version 5.5

– Supports two phase data source:

`com.sybase.jdbc2.jdbc.SybXADataSource`

– Requires JDBC driver files:

`jconn2.jar`

– Requires the following DataStoreHelper class:

`com.ibm.websphere.rsadapter.SybaseDataStoreHelper`

– Requires a valid authentication alias.

– Requires the following properties:

- **serverName** - The name of the database server. Example: *myserver.mydomain.com*
- **databaseName** - The name of the database from which the data source obtains connections. Example: *Sample*.
- **portNumber** - The TCP/IP port number through which all communications to the server take place. Example: *5000*.
- **connectionProperties** - A custom property required for applications containing EJB 2.0 enterprise beans. Value: `SELECT_OPENS_CURSOR=true` (Type: java.lang.String)

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

“Configuring a JDBC provider using the administrative console”
To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

“Configuring a data source using the administrative console” on page 1459
Application components use a data source to access connection instances to a relational database.

“Creating and configuring a JDBC provider and data source using the Java Management Extensions API” on page 1479
If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

Configuring a JDBC provider using the administrative console

To create connections between an application and a relational database, the application server uses the driver implementation classes that are encapsulated by the JDBC provider.

Before you begin

Each JDBC provider is essentially an object that represents vendor-specific JDBC driver classes to the application server, for establishing access to that particular vendor database. JDBC providers are prerequisites for data sources, which supply applications with the physical connections to a database. Consult the JDBC provider table to identify the appropriate JDBC provider for your database and application requirements.

About this task

Configure at least one JDBC provider for each database server that you plan to use at a particular scope within your application server environment.

1. Open the administrative console.
2. Click **Resources** → **JDBC** → **JDBC Providers**.
3. Select the scope at which applications can use the JDBC provider. The scope that you select becomes the scope of any data source that you associate with this provider. You can choose a cell, node, cluster, or server. For more information on scope and how it can affect resources, see the information center topic on administrative scope settings.
4. Click **New**. This action causes the **Create a new JDBC Provider** wizard to launch.
5. Use the first drop-down list to select the database type of the JDBC provider that you need to create.

   Note: Select **User-Defined** for your database type if you encounter either of the following scenarios:
   - You do not see your database type.
   - You cannot select the JDBC provider type that you need in the next step.

   The user-defined selection triggers the wizard panel to display your provider type as a user-defined JDBC provider, and your implementation type as user-defined. Consult your database documentation for the JDBC driver class files, data source properties, and so on that are required for your user-defined provider. You must supply this information on the next two panels:
   - database class path
   - database-specific properties

6. Select your JDBC provider type if it is displayed in the second drop-down list. Select **Show Deprecated** to trigger the display of both current and deprecated providers. If you cannot find your provider in this expanded list, then select **User-Defined** from the previous list of database types.
7. From the third drop-down list, select the implementation type that is necessary for your application. If your application does not require that connections support two-phase commit transactions, choose **Connection Pool Data Source**. Choose **XA Data Source**, however, if your application requires connections that support two-phase commit transactions. Applications that use this data source configuration have the benefit of container-managed transaction recovery.

   After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.
8. Click **Next** to see the **Enter database class path information** wizard panel.
9. In the Class path field, type the full path location of the database JDBC driver class files. Your class path information becomes the value of the WebSphere environment variable that is displayed on this panel, in the form of ${*DATABASE*_JDBC_DRIVER_PATH}. The application server uses the variable to define your JDBC provider; this practice eliminates the need to specify static JDBC class paths for individual applications. Remember that if you do not provide the full, correct JDBC driver class path for the variable, your data source ultimately fails. If the field already displays a fully qualified class path, you can accept that variable definition by completing the rest of this wizard panel and clicking **Next**.

   Note: The application server supports multiple versions of the selected JDBC driver for the DataDirect ConnectJDBC type 4 driver for MS SQL Server. Each version of the JDBC driver has a unique class path. Select the appropriate version of the JDBC driver so the class path is populated correctly.
10. Use the Native library path field to specify additional class files that your JDBC driver might require to function properly on your application server platform. Type the full directory path name of these class files.
11. Click **Next** to see a summary of your JDBC provider settings.
12. Click **Finish** if you are satisfied with the JDBC provider configuration. You now see the JDBC provider collection panel, which displays your new JDBC provider in a table along with other providers that are configured for the same scope.

What to do next

The next step is to create a data source to associate with your JDBC provider. For detailed information, see the information center topic on configuring a data source using the administrative console.

JDBC provider collection:

Use this page to view JDBC providers. The JDBC provider object encapsulates the specific JDBC driver implementation class for the data sources that you define and associate with the provider.

To view this administrative console page, click **Resources** > **JDBC** > **JDBC providers** .

Name:

Specifies a text identifier for this provider.

For example, this field can be *DB2 JDBC Provider (XA)*.

Data type String

Scope:

Specifies the scope of the JDBC provider; if you use any scope other than the default of *Node*, the provider might not be available in other scope contexts. Data sources that are created with this JDBC provider inherit this scope.

Description:

Specifies a text string describing this provider.

Data type String

JDBC provider settings:

Use this page to modify the settings for a JDBC provider.

To view this administrative console page, click **Resources** → **JDBC** → **JDBC providers** → ***JDBC_provider***.

Note: If you use this page to modify the class path or native library path of an existing JDBC provider: After you apply and save the new settings, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Scope:

Specifies the scope of the JDBC provider; data sources that are created with this JDBC provider inherit this scope.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a text description for the resource provider.

Data type String

Class path:

Specifies a list of paths or JAR file names which together form the location for the resource provider classes.

For example:

- *QIBM/ProdData/Java400/ext/db2_classes.jar* for iSeries platforms.

Class path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Class paths contain variable (symbolic) names which you can substitute using a variable map. Check the driver installation notes for the specific required JAR file names.

Data type String

Native Library Path:

Specifies a list of paths that forms the location for the resource provider native libraries.

Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths can contain variable (symbolic) names which you can substitute using a variable map.

Data type String

Isolate this resource provider:

Specifies that this resource provider will be loaded in its own class loader. This allows different versions or implementations of the same resource provider to be loaded in the same Java Virtual Machine. Give each version of the resource provider a unique class path that is appropriate for that version or implementation.

Note: Be aware of the following:

- You cannot isolate a resource provider if you specify a native library path. The Application Server will define a value for the native library path for some JDBC providers; this behavior is intended to help you configure your provider when a native library path is necessary. If you do not require the native library path, delete the value, and you will be able to select the option to isolate the resource provider.
- If you are running a mixed cell environment, the application server will remove any isolated JDBC providers from nodes that are running at versions earlier than 7.0 if the provider is scoped for a version 7.0 cell, and you have not migrated the provider from an older release. If you want to use isolated resources at the cell level, do not use the resources in nodes that are running at versions earlier than 7.0. Define a resource at the node level, or avoid using the resource in nodes that are earlier than version 7.0, because this will result in a ″Naming not found″ exception when the application server attempts to perform a lookup on an isolated resource at the cell level.

Implementation class name:

Specifies the Java class name of the JDBC driver implementation.

This class is available in the driver file mentioned in the class path description above.

For example, *com.ibm.db2.jdbc.app.UDBXADataSource* for iSeries platforms.

Note: If you modify the implementation class name of the JDBC provider after you have created the provider, you might disconnect the provider from the template used to create it. As a result, data sources created from this JDBC provider do not have an associated template; you must manually configure a working data source through setting custom properties.

Data type String

JDBC provider summary:

JDBC providers are prerequisites for data sources, which supply applications with the physical connections to a database.

Use these tables for quick reference on database-specific JDBC providers.

Apache Derby		
JDBC provider	**Transaction support**	**Version and other considerations**
Derby JDBC Provider	One-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.0.2 and later • Not for use in clustered environment: accessible from a single JVM only
Derby JDBC Provider (XA)	One and two-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.0.2 and later • Not for use in clustered environment: accessible from a single JVM only
Derby JDBC Provider 40	One-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources
Derby JDBC Provider 40 (XA)	One and two-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources
Derby Network Server using Derby Client	One-phase	• Does not support Version 4.0 data sources. • Configurable only in nodes at version 6.1 and later • *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs • Only for use with Apache Derby databases that run on the same node as the application server
Derby Network Server using Derby Client (XA)	One and two-phase	• Does not support Version 4 data sources • Configurable only in nodes at version 6.1 and later • *Can be used in clustered environment:* a database instance can be accessed by multiple JVMs • Only for use with Apache Derby databases that run on the same node as the application server
Derby Network Server using Derby Client 40	One-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources

Apache Derby		
JDBC provider	**Transaction support**	**Version and other considerations**
Derby Network Server using Derby Client 40 (XA)	One and two-phase	• Configurable only in nodes at version 7.0 and later • Does not support Version 4 data sources

DB2 on AIX, HP-UX, Linux, Solaris, and Windows systems		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 Using IBM JCC Driver	One-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Using IBM JCC Driver (XA)	One and two-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Universal JDBC Provider	One-phase	N/A
DB2 Universal JDBC Provider (XA)	One and two-phase	N/A

DB2 UDB for iSeries		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 UDB for iSeries (Native)	One-phase	Recommended when you run the application server on iSeries.
DB2 UDB for iSeries (Native XA)	One and two-phase	Recommended when you run the application server on iSeries.
DB2 UDB for iSeries (Toolbox)	One-phase	N/A
DB2 UDB for iSeries (Toolbox XA)	One and two-phase	N/A

DB2 on z/OS		
JDBC provider	**Transaction support**	**Version and other considerations**
DB2 Using IBM JCC Driver	One-phase	• Configurable in version 7.0 and later nodes.
DB2 Using IBM JCC Driver (XA)	One and two-phase	• Configurable in nodes that are at version 7.0 and later.
DB2 Universal JDBC Provider	One-phase when connecting to the application server that is on AIX, HP-UX, Linux, Solaris, and Windows systems	
DB2 Universal JDBC Provider (XA)	One and two-phase	

Informix		
JDBC provider	**Transaction support**	**Version and other considerations**
Informix Using IBM JCC Driver	One phase	This provider is configurable in nodes that are at version 7.0 and later.
Informix Using IBM JCC Driver (XA)	One and two-phase	This provider is configurable in nodes that are at version 7.0 and later.
Informix JDBC Provider	One-phase	N/A
Informix JDBC Provider (XA)	One and two-phase	N/A

Microsoft SQL Server		
JDBC provider	**Transaction support**	**Version and other considerations**
Microsoft SQL Server JDBC Driver	One-phase	N/A
Microsoft SQL Server JDBC Driver (XA)	One and two-phase	N/A
DataDirect ConnectJDBC Provider, type 4 driver, for Microsoft SQL Server	One-phase	N/A
DataDirect ConnectJDBC Provider, type 4 driver, for Microsoft SQL Server (XA)	One and two-phase	N/A

Oracle		
JDBC provider	**Transaction support**	**Version and other considerations**
Oracle JDBC Provider	One-phase	N/A
Oracle JDBC Provider (XA)	One and two-phase	N/A

Sybase		
JDBC provider	**Transaction support**	**Version and other considerations**
Sybase JDBC 3 Driver	One-phase	jConnect v6.05
Sybase JDBC 3 Driver (XA)	One and two-phase	jConnect v6.05
Sybase JDBC 2 Driver	One-phase	jConnect v5.5
Sybase JDBC 2 Driver (XA)	One and two-phase	jConnect v5.5

Configuring a data source using the administrative console

Application components use a data source to access connection instances to a relational database.

Before you begin

The application server supports two different versions of data source. Determine the data source for your environment according to the enterprise bean and servlet specification levels that are the basis of your applications:

- *Data sources (WebSphere Application Server Version 4)* are for use with the Enterprise JavaBeans (EJB) 1.0 specification and the Java Servlet 2.2 specification.
- Data sources of the latest standard version are for use with applications that implement the more advanced releases of these specifications.

About this task

When you create a data source, you associate it with a JDBC provider that is configured for access to a specific vendor database. The application server requires both objects for your applications to make calls to that particular database and receive data from it. The data source provides connection management capabilities that physically make possible these exchanges between your applications and the database.

1. Open the administrative console.
2. Access the necessary console panel. Use one of the following paths:
   - Click **Resources** → **JDBC** → **Data sources**.
   - Click **Resources** → **JDBC** → **Data sources (WebSphere Application Server Version 4)**

- Click **Resources** → **JDBC** → **JDBC providers** → ***jdbc_provider*** → **Data sources**
- Click **Resources** → **JDBC** → **JDBC providers** → ***jdbc_provider*** → **Data sources (WebSphere Application Server Version 4)**.

3. Select the scope at which applications can use the data source. You can choose a cell, node, cluster, or server. For more information, see the topic on scope settings.

   Note: From this point onward, the steps for creating WebSphere Application Server Version 4 data sources differ from the steps for creating data sources of the latest standard version. To configure a Version 4 data source:
   - Click **New** to proceed to the console panel for defining required properties.
   - On this properties panel specify values for the fields that are grouped under the heading **Configuration**. The application server requires these properties to implement your JDBC driver classes.
   - Save your configuration by clicking **OK**. You are now finished with the primary data source configuration tasks.
   - Define other properties that your database vendor might require, or offer as options, for using the JDBC driver. The application server calls them custom properties, and requires that you set them on the data source. Begin by clicking the **Custom Properties** link that is now displayed on the administrative console panel. Consult your database documentation to learn about these required and optional properties.
4. Click **New**. This action causes the **Create a data source** wizard to launch and display the **Enter basic data source information** panel. The first field is the scope field, which is read-only. This field displays your previous scope selection.
5. Type a data source name in the **Data source name** field. This name identifies the data source for administrative purposes only.
6. Type a Java Naming and Directory Interface (JNDI) name in the **JNDI name** field. The application server uses the JNDI name to bind resource references for an application to this data source. Follow these requirements when you specify JNDI names:
   - Do not assign duplicate JNDI names across different resource types, such as data sources versus J2C connection factories or JMS connection factories.
   - Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.

   For more information on JNDI, consult the topic on naming.
7. Click **Next** to see the **Select JDBC provider** panel . The **Select JDBC provider** panel is skipped if you do not have any JDBC providers that are configured at the current scope.
8. Select an existing JDBC provider, or create a new provider.
   - Select an existing JDBC provider.
     a. Click **Select an existing JDBC provider**.
     b. Select a JDBC driver from the drop-down list.
     c. Click **Next**. You now see the panel entitled **Enter database specific properties for the data source**.
   - Create a new JDBC provider.
     a. Click **Create new JDBC provider**.
     b. Click **Next** to see the **Create JDBC provider** panel.
     c. Use the first drop-down list to select the database type of the JDBC provider that you need to create.

        Note: Select **User-Defined** for your database type if you encounter either of the following scenarios:
        - You do not see your database type.
        - You cannot select the JDBC provider type that you need in the next step.

The user-defined selection triggers the wizard panel to display your provider type as a User-defined JDBC provider, and your implementation type as User-defined. Consult your database documentation for the JDBC driver class files, data source properties, and so on that are required for your user-defined provider. You must supply this information on the next two wizard panels:

– database class path information

– database-specific properties

d. If the JDBC provider type is displayed in the second drop-down list, select your JDBC provider type. Select **Show Deprecated** to trigger the display of both current and deprecated providers. If you cannot find your provider in this expanded list, then select **User-Defined** from the previous list of database types.

e. From the third drop-down list, select the implementation type that is necessary for your application. If your application does not require that connections support two-phase commit transactions, choose **Connection Pool Data Source**. Choose **XA Data Source**, however, if your application requires connections that support two-phase commit transactions. Applications that use this data source configuration have the benefit of container-managed transaction recovery.

After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.

f. Click **Next** after you have defined your database type, provider type, and implementation type. Now you see the wizard panel Enter database class path information.

g. In the class path field, type the full path location of the database JDBC driver class files. Your class path information becomes the value of the WebSphere environment variable that is displayed on this panel, in the form of ${*DATABASE*_JDBC_DRIVER_PATH}. The application server uses the variable to define your JDBC provider; this practice eliminates the need to specify static JDBC class paths for individual applications. Remember that if you do not provide the full, correct JDBC driver class path for the variable, your data source ultimately fails. If the field already displays a fully qualified class path, you can accept that variable definition by completing the rest of this wizard panel and clicking **Next**.

h. Use the **Native library path** field to specify additional class files that your JDBC driver might require to function properly on your application server platform. Type the full directory path name of these class files.

i. Click **Next**. You now see the **Enter database specific properties for the data source** panel.

9. Complete all of the fields on the **Enter database specific properties for the data source** panel.

- Click **Use this data source in container managed persistence (CMP)** if container managed persistence (CMP) enterprise beans must access this data source.
- Any other property fields that are displayed on this wizard panel are specific to your database type. Consult the article “Data source minimum required settings, by vendor” on page 1433 for information on these property settings. The article addresses both current and deprecated JDBC providers that are pre-defined in the application server.

Note: This wizard panel does not display additional property fields for data sources that correspond with your user-defined JDBC providers. However, from the JDBC driver class files that you installed, The application server can generally extract the necessary data source property names. The application server defines them as data source custom properties, displays them on a custom properties console panel, and assigns them default values. Consult your database documentation about setting these properties and any other requirements for your user-defined data source. After you create the data source, navigate to the corresponding custom properties collection panel in the administrative console by clicking **Data sources** → ***data_source*** → **Custom properties**. Review the property default values and modify them if necessary.

10. Optional: Configure the security aliases for the data source. You can select none for any of the authentication methods, or choose one of the following types:
    - **Component-managed authentication alias** - specifies an authentication alias to use when the component resource reference res-auth value is Application. To define a new alias, navigate to **Related Items** → **J2EE Connector Architecture (J2C) authentication data entries**. A component-managed alias represents a combination of ID and password that is specified in an application for data source authentication. Therefore, the alias that you set on the data source must be identical to the alias in the application code.
      a. Use the drop-down list to select an existing component-managed authentication alias.
      b. To create a new alias, click the links that are provided. This action closes the data source wizard and triggers the administrative console to display the J2C authentication data panel. Click **New** to define a new alias. Click **OK** to save your settings and view the new alias on the J2C authentication data panel. Restart the data source wizard by navigating back to the data source collection panel, selecting the appropriate scope, and clicking **New**.

      For more information on Java 2 Connector (J2C) security, see the topic on managing Java 2 Connector Architecture authentication data entries.
    - **Mapping-configuration alias** - will be used only in the absence of a login configuration on the component resource reference. The specification of a login configuration and the associated properties on the component resource reference is the preferred way to define the authentication strategy when the res-auth value is set to Container. If you specify the DefaultPrincipalMapping login configuration, the associated property will be a JAAS - J2C authentication data entry alias.
    - **Container-managed authentication alias** - will be used only in the absence of a login configuration on the component resource reference. The specification of a login configuration and the associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is set to Container.

    Note: If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.
11. Click **Next** to view the **Summary** panel, and review any information for the data source. If any information is not correct, you can click **Previous** to go back and correct it.
12. Click **Finish** to save the configuration and exit the wizard. You now see the **Data sources** panel, which displays your new configuration in a table along with other data sources that are configured for the same scope.

What to do next

You can override the default values for some data source properties. You can also configure additional properties that your database vendor might require or offers as options. Consult your database documentation about these settings. The following topics in this information center inform you of how to use the administrative console to assign the property values:

- “Connection pool settings” on page 1400
- “Tuning connection pools”
- “WebSphere Application Server data source properties” on page 1469
- “Java EE resource provider or connection factory custom properties collection” on page 1476

Tuning connection pools:

Using connection pools helps to both alleviate connection management overhead and decrease development tasks for data access. Each time an application attempts to access a backend store (such as

a database), it requires resources to create, maintain, and release a connection to that datastore. To mitigate the strain this process can place on overall application resources, the application server enables administrators to establish a pool of backend connections that applications can share on an application server. Connection pooling spreads the connection overhead across several user requests, thereby conserving application resources for future requests.

About this task

Connection pooling can improve the response time of any application that requires connections, especially Web-based applications. When a user makes a request over the Web to a resource, the resource accesses a data source. Because users connect and disconnect frequently with applications on the Internet, the application requests for data access can surge to considerable volume. Consequently, the total overhead for a datastore can become high for Web-based applications quickly, and performance deteriorates. When connection pooling capabilities are used, however, Web applications can realize performance improvements of up to 20 times the normal results.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

- **Prevent a connection deadlock.** Deadlock can occur if the application requires more than one concurrent connection per thread, and the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:
  - Each thread has its first database connection, and all are in use.
  - Each thread is waiting for a second database connection, and none would become available since all threads are blocked.

  To prevent the deadlock in this case, increase the maximum connections value for the database connection pool by at least one. This will allow at least one of the waiting threads to obtain a second database connection and avoid a deadlock scenario.

  For general prevention of connection deadlock, code your applications to use only one connection per thread. If you code the application to require *C* concurrent database connections per thread, the connection pool must support at least the following number of connections, where *T* is the maximum number of threads:

```
T * (C - 1) + 1
```

  The connection pool settings are directly related to the number of connections that the database server is configured to support. If you increase the maximum number of connections in the pool and the corresponding settings in the database are not increased accordingly, the application might fail. The resulting SQL exception errors would be displayed in the following location(s):
  - the stderr.log file
- **Disable connection pooling.**
  - For relational resource adapters (RRAs), add the `disableWASConnectionPooling` custom property for your data sources.
    1. Click **JDBC** > **Data sources**.
    2. Click on the name of the data source that you want to configure.
    3. Click **Custom properties** under the **Additional Properties** heading.
    4. Click **New**.

5. Complete the required fields with the following information:

Table 10. Custom property panel

Name	Value
disableWASConnectionPooling	true

– For other resource adapters, consult with the binding specifications for that resource adapter to configure your applications to disable connection pooling.
  1. Programmatically disable connection pooling through the resource adapter.
  2. The application server leverages the following code to detect the javax.resource.NotSupportedException exception and disable connection pooling:

```
_managedFactory.matchManagedConnections(s,subject,cri); // 169059 174269     }
 catch(javax.resource.NotSupportedException e){
```

- **Enable deferred enlistment.** In the application server environment, deferred enlistment refers to the technique in which the application server waits until the connection is used before the connection is enlisted in the application's unit of work (UOW) scope.

  Consider the following illustration of deferred enlistment:

  – An application component that uses deferred enlistment calls the `getConnection` method from within a global transaction.
  – The application component does not immediately use the connection.
  – When the application issues the call for initial use of the connection, the transaction manager intercepts the call.
  – The transaction manager enlists the XA resource for the connection and calls the `XAResource.start` method.
  – The connection manager associated with the XA resource sends the call to the database.

  Given the same scenario, but the application component does not use deferred enlistment, the component container immediately enlists the connection in the transaction. Thus the application server incurs, for no purpose, an additional load of all of the overhead associated with that transaction. For XA connections, this overhead includes the two phase commit (2PC) protocol to the resource manager.

  Deferred enlistment offers better performance in the case where a connection is obtained, but not used, within the UOW scope. The technique saves the cost of transaction participation until the UOW in which participation must occur.

  Check with your resource adapter provider if you need to know if the resource adapter provides this functionality. The application server relational resource adapter automatically supports deferred enlistment.

  Incorporating deferred enlistment in your code:

  The Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) Version 1.5 specification calls the deferred enlistment technique lazy transaction enlistment optimization. This support comes through a marker interface (LazyEnlistableManagedConnection) and a new method on the connection manager (LazyEnlistableConnectionManager()):

```
package javax.resource.spi; import javax.resource.ResourceException; import
 javax.transaction.xa.Xid;  interface LazyEnlistableConnectionManager { // application server     void
 lazyEnlist(ManagedConnection) throws ResourceException; }  interface LazyEnlistableManagedConnection { // resource adapter }
```

Disabling statement pooling:

Disable statement pooling performing Data Definition Language (DDL) operations, which might not be compatible with statement pooling.

About this task

DDL operations, such as dropping and recreating tables, are not compatible with statement pooling when using the IBM Informix database. DDL operations invalidate the pooled statements or cause them to produce unexpected results.

Complete the following steps to use the administrative console to disable statement pooling. Statement pooling is disabled by specifying a value of zero for the statement cache size setting for your datasource.

1. From the administrative console, select **Resources** → **JDBC** → **Data sources** → ***datasource_name***.
2. Under Additional properties, click **WebSphere Application Server data source properties**.
3. Enter a value of zero (0) in the Statement cache size field.
4. Click **OK**, and then click **Save**.

Data source collection:

Use this page to view configured data sources, which are the resources that provide connections to your relational database.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC** > **Data sources**
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**

Note: If you are using the Enterprise JavaBean (EJB) 1.0 specification and the Java Servlet 2.2 specification, you must use the **Data sources (WebSphere Application Server Version 4)** console page.

Name:

Specifies the display name of this data source.

Click a data source name to edit the data source configuration settings.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for this data source.

Data type String

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the appropriate classes.

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a string that you can use to classify or group a data source.

Data type String

Data source settings:

Use this panel to edit the properties of a data source.

You can access this administrative console page in one of two ways:

- **Resources** → **JDBC** → **Data sources** → ***data_source***
- **Resources** → **JDBC** → **JDBC providers** → ***JDBC_provider*** → **Data sources** → ***data_source***

Note: If your application uses an Enterprise JavaBean (EJB) 1.1 or a Java Servlet 2.2 module, use the **Data sources (WebSphere Application Server V4)** → ***data_source*** console page.

Test connection:

Activates the test connection service for validating application connections to the data source.

Before you click **Test connection**, set your data source properties and click **Apply**.

Scope:

Specifies the scope of the JDBC provider that supports this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the driver implementation classes to support this data source.

Name:

Specifies the display name for the data source.

Valid characters for this name include letters and numbers, but NOT most of the special characters. For example you can set this field to *Test Data Source*. But any name starting with a period (•) or containing special characters (\ / , : ; ″ * ? < > | = + & % ’ ` @) is not a valid name.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects.

There are many naming and directory service implementations, and the interfaces to them vary. JNDI provides a common interface that is used to access the various naming and directory services.

For example, you can use the name *jdbc/markSection*.

If you leave this field blank a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of *jdbc/markSection*.

After you set this value, save it, and restart the server, you can see this string when you run the dump name space tool.

Data type String

Use this data source in container-managed persistence (CMP):

Specifies if this data source is used for container-managed persistence of enterprise beans.

This option triggers creation of a CMP connection factory, which corresponds to this data source, for the relational resource adapter.

Data type Boolean
Default True (enabled)

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string you can use to classify or group the resource.

Data type String

Data store helper class name:

Specifies the name of the DataStoreHelper implementation class that extends the capabilities of your selected JDBC driver implementation class to perform database-specific functions.

The application server provides a set of DataStoreHelper implementation classes for each of the JDBC provider drivers it supports. These implementation classes are in the package com.ibm.websphere.rsadapter. For example, if your JDBC provider is DB2, then your default DataStoreHelper class is com.ibm.websphere.rsadapter.DB2DataStoreHelper. The administrative console page you are viewing, however, might make multiple DataStoreHelper class names available to you in a drop-down list; be sure to select the one required by your database configuration. Otherwise, your application might not work correctly. If you want to use a DataStoreHelper other than those displayed in the drop-down list, select **Specify a user-defined DataStoreHelper**, and type a fully qualified class name. Refer to the information center for instructions on creating a custom DataStoreHelper class.

Data type Drop-down list or string (if **user-defined DataStoreHelper** is selected)

Component-managed authentication alias:

This alias is used for database authentication at run time.

The **Component-managed Authentication Alias** is only used when the application resource reference is using *res-auth = Application*.

If your database does not support *user ID* and *password*, then do not set the alias in the component-managed authentication alias or container-managed authentication alias fields. Otherwise, you see the warning message in the system log to indicate that the user and password are not valid properties. This message is only a warning message; the data source is still created successfully.

If you do not set an alias (component-managed or otherwise), and your database requires the user ID and password to get a connection, then you receive an exception during run time.

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Drop-down list

Authentication alias for XA recovery:

This field is used to specify the authentication alias that should be used during XA recovery processing. If this alias name is changed after a server failure, the subsequent XA recovery processing will use the original setting that was in effect before the failure.

If the resource adapter does not support XA transactions, then this field will not be displayed. The default value will come from the selected alias for application authentication (if specified).

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Drop-down list

Container-managed authentication alias:

Specifies authentication data, which is a JAAS - J2C authentication data entry, for container-managed signon to the resource.

Select an alias from the list.

To define a new alias that is not displayed in the list:

1. Click **Apply**. Under Related Items, you now see a listing for J2EE Connector Architecture (J2C) authentication data entries.
2. Click **J2EE Connector Architecture (J2C) authentication data entries**.
3. Click **New**.
4. Define an alias.

5. Click **OK**. The console now displays an alias collection page. This page contains a table that lists all of your configured aliases. Before the table, this page also displays the name of your connection factory.
6. Click the name of your J2C connection factory. You now see the configuration page for the connection factory.
7. Select the new alias in the Container-managed authentication alias list.
8. Click **Apply**.

If you have defined security domains in the application server, you can click **Browse...** to select an authentication alias for the resource that you are configuring. Security domains allow you to isolate authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Drop-down list

Mapping-configuration alias:

Specifies the authentication alias for the Java Authentication and Authorization Service (JAAS) mapping configuration that is used by this connection factory.

Click **Security** → **Global security**. In the **Authentication** section, click **Java Authentication and Authorization Service** → **Application logins**, and select an alias from the table.

The **DefaultPrincipalMapping** JAAS configuration maps the authentication alias to the user ID and password. You can define and use other mapping configurations.

Data type Drop-down list

Common and required data source properties: These properties are specific to the data source that corresponds to your selected JDBC provider. They are either required by the data source, or are especially useful for the data source. You can find a complete list of the properties required for all supported JDBC providers in the information center.

WebSphere Application Server data source properties:

Use this page to set advanced data source properties in the application server. These properties activate and configure services that the application server applies to data sources to customize connections within an application server. These properties do not affect connections within the database.

To access this administrative console page complete one of the following paths:

- **Resources** → **JDBC** → **Data sources** → ***data_source*** → **WebSphere Application Server data source properties**
- **Resources** → **JDBC** → **JDBC providers** → ***JDBC_provider*** → **Data sources** → ***data_source*** → **WebSphere Application Server data source properties**
- **Applications** → **Application Types** → **Websphere enterprise applications** → ***application_name*** → **Application scoped resources** → ***data_source*** → **WebSphere Application Server data source properties**.

Statement cache size:

Specifies the number of statements that can be cached per connection. The application server caches a statement after you close that statement.

The WebSphere Application Server data source optimizes the processing of prepared statements and callable statements by caching those statements that are not used in an active connection. Both statement types help maximize the performance of transactions between your application and datastore.

- A prepared statement is a precompiled SQL statement that is stored in a PreparedStatement object. The application server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time.
- A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the CallableStatement object. The application server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

If the statement cache is not large enough, useful entries are discarded to make room for new entries. To determine the highest value for your cache size to avoid any cache discards, add the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. Setting the cache size to this value means you never have cache discards. In general, configure a larger cache for applications with a greater number of statements.

You can also use the Tivoli Performance Viewer to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings.

Note: The higher the statement cache, the more system resources are delayed. Therefore, if you set the number too high, then you might lack resources because your system cannot open multiple prepared statements.

If there is a particular statement that you do not want the application server to cache, configure the statement's poolability hint to false. The application server does not cache a statement if the poolability hint is set to false. The application specifies the statement poolability hints at run time.

In test applications, tuning the statement cache improves throughput from 10% to 20%. However, because of potential resource limitations, this might not always be possible.

Data type	Integer
Default	Default values depend on the database. Typically, this value is 10. For Informix versions 7.3, 9.2, 9.3, and 9.4, without the respective latest fixes, the default value must be 0. A default value of 0 means that there is no cache statement.

Enable multithreaded access detection: When you check this option, the application server detects the existence of access by multiple threads.

Enable database reauthentication: Indicates that the exact match on connections retrieved out of the application server's connection pool (the connection pool search criteria does not include a user name and password) cannot exist. Instead, the connection reauthentication is done in the doConnectionSetupPerTransaction() of the DataStoreHelper class. The application server does not provide a connection reauthentication implementation at run time. Therefore, when you check this box, you must extend the DataStoreHelper class to provide implementation of the doConnectionSetupPerTransaction() method where the reauthentication occurs. If you do not complete this process, the application server might return unusable connections. For more information, refer to the API documentation for the com.ibm.websphere.rsadapter.DataStoreHelper#doConnectionSetupPerTransaction method.

Connection reauthentication can help improve performance by reducing the overhead of opening and closing connections, particularly for applications that frequently request connections with different user names and passwords.

Note: You cannot enable database reauthentication if you select TrustedConnectionMapping for the mapping configuration alias.

Enable JMS one-phase optimization support: When you check this option, the application server allows Java messaging service (JMS) to get optimized connections from this data source. This property prevents Java database connectivity (JDBC) applications from sharing connections with container-managed persistence (CMP) applications.

Manage cached handles: Specifies whether the container tracks cached handles, which are connection handles that an application component holds active across transaction and method boundaries. You can use this property to debug connection problems, but tracking handles can cause large performance overhead during run time.

If the **Manage cached handles** property is selected in the administrative console, and you deselect it, the field will no longer be visible for resources that are at Version 7.0 of the application server. This field is only displayed if the manageCachedHandles property is set to `true` in the resources.xml file. To make the field available, change the value for the manageCachedHandles entry from `false` to `true` in the resources.xml file, or enter the following Jython command from the wsadmin tool:

```
AdminConfig.modify(myDataSourceVariable, '[[manageCachedHandles "true"]]')
```

Note: For any resources that are running at Version 6.x of the application server, the **Manage cached handles** property will always be visible. For example, if you have a node that is at Version 6.1, the entry in the resources.xml file will not affect how the field is displayed in the administrative console.

For an alternative method of debugging problems, use the multi-thread and cross-component diagnostic alerts to detect violations in the Java Connectivity Architecture (JCA) programming model. To enable these alerts, select those options from **Servers** → **Application servers** → ***application_server*** → **Performance** → **Performance and Diagnostic Advisor Configuration** → **Performance and Diagnostic Advice configuration** panel. These alerts force the connection manager to manage cached handles, detect the connection conditions, and send alerts.

Note: For these alerts to be active you must also select **Enable Performance and Diagnostic Advisor Framework (Runtime Performance Advisor)** from the **Servers** → **Application servers** → ***application_server*** → **Performance** → **Performance and Diagnostic Advisor Configuration** panel.

Log missing transaction context: Specifies whether the container issues an entry to the activity log when an application obtains a connection without a transaction context. These are exceptions to the Java Platform, Enterprise Edition (Java EE) programming model connection requirements.

Non-transactional data source: Specifies that the application server will not enlist the connections from this data source in global or local transactions. Applications must explicitly call setAutoCommit(false) on the connection if they want to start a local transaction on the connection, and they must commit or rollback the transaction that they started.

Note: Set this property to true in rare circumstances, but the Java Persistence API (JPA) requires both JTA and non-JTA data sources.

Use WebSphere Application Server exception checking model:

Specifies that the application server uses the error mapping facility that is defined in the data store helper to identify errors. The application server does not replace exceptions that are thrown by the JDBC driver with exceptions that are defined in the error map of the data store helper.

Use WebSphere Application Server exception mapping model:

Specifies that the application server uses the error mapping facility that is defined in the data store helper to identify errors, and the application server will replace the exceptions that are thrown by the JDBC driver with exceptions that are defined in the error map of the data store helper.

Note: This error detection model functions with JDBC Version 3.0 and earlier.

Validate new connections: Specifies whether the connection manager tests newly created connections to the database.

Number of retries: Specifies the number of times you want to retry making the initial connection to a database after the first pretest operation fails.

Retry interval: If you select **Validate new connections**, this option specifies the length of time, in seconds, that the application server waits before retrying to make a connection if the initial attempt fails.

Validate existing pooled connections: Specifies whether the connection manager tests the validity of pooled connections before returning them to applications.

Retry interval: If you select **Pretest existing pooled connections**, this option specifies the length of time, in seconds, to allot to the JDBC driver for validating a connection.

Validation by JDBC driver:

Specifies that the application server will use the JDBC driver to validate the connections. The JDBC provider must support JDBC 4.0 or greater to use this option.

Timeout: Specifies the timeout in seconds for testing connections (either new or pooled by the application server) to the database. If the timeout expires before validating then the connection is considered unusable. If retries are configured, the full value of the timeout applies to each retry. A value of 0 indicates the JDBC driver does not impose a timeout on validation attempts.

Note: This option is only available for JDBC drivers that are JDBC 4.0 compliant.

Validation by SQL string (deprecated):

Specifies an SQL statement that the application server sends to the database to test the connection. Use a query that is likely to have low impact on performance.

Optimize for get/use/close/connection pattern with heterogenous pooling:

Specifies that the application server will use the get/use/close/connection pattern. This allows connection pooling for the application server to share connections that are in the same transaction. This optimization pattern allows one connection to be shared during a transaction even when connections use different connection properties.

The heterogeneous pooling feature allows you to extend the data source definition so that you can specify different custom properties or allow applications to override non-core properties for the data source.

Note: This field is only available for DB2 data sources.

Retry interval for client reroute: Specifies the amount of time, in seconds, between retries for automatic client reroute.

Note: This field is only available for DB2 data sources.

Maximum retries for client reroute: Specifies the maximum number of connection retries that are attempted by the automatic client reroute function if the primary connection to the server fails. The property is only used when **Retry interval for client reroute** is set.

Note: This field is only available for DB2 data sources.

Alternate server names: Specifies the list of alternate server name or names for the DB2 server. If more than one alternate server name is specified, the names must be separated by commas. For example:

```
host1,host2
```

Note: This field is only available for DB2 data sources.

Alternate port numbers: Specifies the list of alternate server port or ports for the DB2 server. If more than one alternate server port is specified, the ports must be separated by commas. For example:

```
5000,50001
```

Note: This field is only available for DB2 data sources.

Client reroute server list JNDI name: Specifies the JNDI name that is used to bind the DB2 client reroute server list into the JNDI name space. The DB2 database server will use this name to look up the alternate server name list when the alternate server information is not already in memory. This option is not supported for type 2 data sources.

Note: This field is only available for DB2 data sources.

Unbind client reroute list from JNDI: Used with test connection only. When set to true, the Client reroute server list JNDI name will be unbound from the JNDI name space after a test connection is issued.

Note: This field is only available for DB2 data sources.

Data source (WebSphere Application Server V4) collection:

Use this page to view the settings of a WebSphere Application Server Version 4.0 data source.

These data sources use the WebSphere Application Server Version 4.0 Connection Manager architecture. All EJB 1.1 modules must use a WebSphere Application Server Version 4.0 data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC**> **Data sources (WebSphere Application Server V4)**
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources (WebSphere Application Server V4)**

Name:

Specifies a text identifier of the data source.

Data type String

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name of the data source.

Data type String

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that encapsulates the appropriate classes.

Description:

Specifies a text description of the data source.

Data type String

Category:

Specifies a text string that you can use to classify or group the data source.

Data type String

Data source (WebSphere Application Server Version 4) settings:

Use this page to create a Version 4.0 style data source. This data source uses the WebSphere Application Server Version 4.0 connection manager architecture. All of your EJB1.x modules must use this data source.

You can access this administrative console page in one of two ways:

- **Resources** > **JDBC**> **Data sources (WebSphere Application Server V4)** > *data_source*
- **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources (WebSphere Application Server V4)** > *data_source*

Scope:

Specifies the scope of the JDBC provider that encapsulates the driver implementation classes to support this data source. Only applications that are installed within this scope can use this data source.

Provider:

Specifies the JDBC provider that WebSphere Application Server uses for this data source.

The list shows all of the existing JDBC providers that are defined at the relevant scope. Select one from the list if you want to use an existing JDBC provider as Provider.

Create new provider:

Provides the option of configuring a new JDBC provider for the new data source.

Create New Provider is displayed only when you create, rather than edit, a data source.

Clicking **Create New Provider** causes the **Create a new JDBC provider** wizard to launch. Complete all of the wizard panels, then click **Finish**. The administrative console now displays the Data sources (WebSphere Application Server V4) configuration page again, where you see the new JDBC provider name in the Provider field.

Name:

Specifies the display name for the resource.

For example, you can set this field to *Test Data Source*.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name.

Distributed computing environments often employ naming and directory services to obtain shared components and resources. Naming and directory services associate names with locations, services, information, and resources.

Naming services provide name-to-object mappings. Directory services provide information on objects and the search tools required to locate those objects.

There are many naming and directory service implementations, and the interfaces to them vary. JNDI provides a common interface that is used to access the various naming and directory services.

For example, you can use the name *jdbc/markSection*.

If you leave this field blank a JNDI name is generated from the name of the data source. For example, a data source name of markSection generates a JNDI name of *jdbc/markSection*.

After you set this value, save it, and restart the server, you can see this string when you run the dump name space tool.

Data type String

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string that you can use to classify or group the resource.

Data type String

Database name:

Specifies the name of the database that this data source accesses.

For example, you can call the database *SAMPLE*.

Data type String

Default user ID:

Specifies the user name to use for connecting to the database.

For example, you can use the ID *db2admin*.

Data type String

Default password:

Specifies the password used for connecting to the database.

For example, you can use the password *db2admin*.

Data type String

Java EE resource provider or connection factory custom properties collection:

Use this page to view the custom properties of a Java Platform, Enterprise Edition (Java EE) resource provider or connection factory.

You can configure custom property collections for numerous resource types. According to the resource type with which a collection is associated, your ability to add, delete, and modify individual properties and settings varies. Begin the configuration process by clicking on the *Required* field to sort those column values in descending order. All of the required (true) values are then sorted at the beginning of the page. Be sure to set all required properties.

Note: The following list displays three custom properties that are deprecated in WebSphere Application Server Version 6.10. The product now offers these properties as pre-configured options, which are the replacement properties in the following list. To avoid runtime error messages, permanently disable the original custom properties by deleting them from the table on this administrative console page.

- dbFailOverEnabled -- replaced by **Pretest new connection**
- validateNewConnection -- replaced by **Number of retries**
- connRetriesDuringDBFailover -- replaced by **Retry interval**

To set the replacement properties, click **Data sources** > *my_data_source* > **WebSphere Application Server data source properties**.

Name:

Specifies the property name.

You must ensure that the resource provider has the setting for this name.

Data type String

Value:

Specifies the property value.

Data type Variable; see “Custom property settings” for more information.

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies whether this property is required for the resource provider.

Data type Boolean or Check box

Custom property settings:

Use this page to specify the attributes of custom properties that might be required for resource providers and resource factories.

According to the resource type with which a property collection is associated, your ability to modify individual property settings varies. Therefore, consider the following descriptions as a general reference for custom property settings. (The administrative console page that you are using to configure your custom property may only allow you to modify a subset of the following settings.)

Note: The following list displays three custom properties that are deprecated in WebSphere Application Server Version 6.10. The product now offers these properties as pre-configured options, which are the replacement properties in the following list.

- dbFailOverEnabled -- replaced by **Pretest new connection**
- validateNewConnection -- replaced by **Number of retries**
- connRetriesDuringDBFailover -- replaced by **Retry interval**

Set the replacement properties on the WebSphere Application Server data source properties console page. Click **Data sources** > *my_data_source* > **WebSphere Application Server data source properties**.

Required:

Specifies properties that are required for this resource.

Data type Check box

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Determined by the **Type** setting, which you select from a drop-down list. If the type is `java.lang.String` then the value is of type String; if the type is `java.lang.Integer`, then the value is of type Integer; and so on.

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java data type of this property .

There are specific types that are valid:
- java.lang.Boolean
- java.lang.String
- java.lang.Integer
- java.lang.Double
- java.lang.Byte
- java.lang.Short
- java.lang.Long
- java.lang.Float

Data type Drop-down list

Custom Properties (Version 4) collection:

Use this page to view properties for a WebSphere Application Server Version 4.0 data source.

You can access this administrative console page in one of two ways:
- **Resources** > **JDBC** > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties**
- **Resources** > **JDBC** > **JDBC Providers** > *JDBC_provider* > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties**

Name:

Specifies the name of the custom property

Data type String

Value:

Specifies the value of the custom property.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Required:

Specifies properties that are required for this resource.

Data type String

Custom property (Version 4) settings:

Use this page to add properties for a WebSphere Application Server Version 4.0 data source.

To view this administrative console page, click **Resources** > **JDBC** > **JDBC Providers**> *JDBC_provider* > **Data Sources (WebSphere Application Server V4)** > *data_source* > **Custom Properties** > *custom_property*.

Required:

Specifies properties that are required for this resource.

Data type Check box

Name:

Specifies the name associated with this property (PortNumber, ConnectionURL, etc).

Data type String

Value:

Specifies the value associated with this property in this property set.

Data type Integer

Description:

Specifies text to describe any bounds or well-defined values for this property.

Data type String

Type:

Specifies the fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).

Data type String

Creating and configuring a JDBC provider and data source using the Java Management Extensions API

If your application requires access to a JDBC connection pool from a Java 2 Platform, Enterprise Edition (J2EE) 1.3 or 1.4, or Java Platform Enterprise Edition (Java EE) level WebSphere Application Server component, you can create the necessary JDBC provider and data source objects using the Java Management Extensions (JMX) API exclusively.

About this task

Alternatively, you can use the JMX API in combination with the WSadmin - scripting tool.

Note: Use the JMX API to create only data sources for which the product does *not* provide a template. For every JDBC provider WebSphere Application Server supports, the product provides a corresponding data source template. You can create supported providers and associated data sources through the administrative console, or by using the WSadmin - scripting tool. For a complete list of supported JDBC providers (and therefore a complete list of data sources that must be created using a template), refer to the topic “Data source minimum required settings, by vendor” on page 1433.

These steps outline the general procedure for using the JMX API to create a JDBC provider and data source, on WebSphere Application Server running on Windows platforms.

1. Put the appropriate JAR files in your classpath. You need thecom.ibm.ws.admin.client_6.1.0.jar JAR file in your classpath.

   The following command is an example for setting your classpath:

   ```
   set classpath=%classpath%;install_root\runtime\com.ibm.ws.admin.client_6.1.0.jar
   ```

2. Look up the host and get an administration client handle.
3. Get a configuration service handle.
4. Update the `resource.xml` file using the configuration service as desired.
   a. Add a JDBC provider.
   b. Add the data source.
   c. Add the connection factory. This step is necessary only for data sources that must support container-managed persistence.
5. Reload the `resource.xml` file to bind the newly created data source into the JNDI namespace. Perform this step if you want to use the newly created data source right away without restarting the application server.
   a. Locate the DataSourceConfigHelper MBean using the name.
   b. Put together the signature and parameters for the call.
   c. Invoke the reload() call.
6. **Attention:** If you modify the class path or native library path of an existing JDBC provider, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

Example: Using the Java Management Extensions API to create a JDBC driver and data source for a CMP bean:

This code sample demonstrates how to configure a JDBC provider and data source, designate the data source for use with CMP beans, set an authorization alias for the data source, and reload the Mbean to make configuration changes.

```
//
// "This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using, marketing, or
// distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines
// Corp., 2008
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;
```

```
/**
 * Creates a node scoped resource.xml entry for a DB2 XA data source.
 * The data source created is for CMP use.
 *
 * Set the following for the program to run:
 * set classpath=%classpath%;install_root\runtime\com.ibm.ws.admin.client_6.1.0.jar
 * install_root/lib/bootstrap.jar;install_root/lib/j2ee.jar;
 * install_root/plugins/com.ibm.ws.bootstrap_6.1.0.jar;install_root/plugins/com.ibm.ws.emf_2.1.0.jar;
 * install_root/plugins/com.ibm.ws.runtime_6.1.0.jar;install_root/plugins/org.eclipse.emf.common_2.2.1.v200609210005.jar;
 * install_root/plugins/org.eclipse.emf.ecore_2.2.1.v200609210005.jar;
 */
public class CreateDataSourceCMP {

   String dsName = "markSection"; // the display name for the data source,
                                  // the JNDI name, and the connection factory name
   String dbName = "SECTION";     // the database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // user ID
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the database driver



   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceCMP cds = new CreateDataSourceCMP();

      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }


   /**
    * This method creates the data source using JMX.
    * The data source created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted or
    * an application is started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         // Specify the SOAP_CONNECTOR_ADDRESS port of your server configuration.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE, +
          AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
         adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
         AdminClient adminClient = +
              AdminClientFactory.createAdminClient(adminProps);

         // Get the ConfigService implementation.
         com.ibm.websphere.management.configservice.ConfigServiceProxy
           configService =
         new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

         Session session = new Session();

         // Use this group to add to the node scoped resource.xml.
         ObjectName node1 = ConfigServiceHelper.createObjectName(null,"Node",
            null);
         ObjectName[] matches = configService.queryConfigObjects(session, null,
            node1, null);
         node1 = matches[0];     // use the first node found

         // Use this group to add to the server1 scoped resource.xml.
         ObjectName server1 = ConfigServiceHelper.createObjectName(null,
                 "Server", "server1");
         matches = configService.queryConfigObjects(session, null, server1,
                null);
         server1 = matches[0];   // use the first server found

         // Create the JDBCProvider
         String providerName = "DB2 JDBC Provider (XA)";
         System.out.println("Creating JDBCProvider " + providerName );

         // Prepare the attribute list
         AttributeList provAttrs = new AttributeList();
```

```
provAttrs.add(new Attribute("name", providerName));
provAttrs.add(new Attribute("implementationClassName",
        "com.ibm.db2.jdbc.DB2XADataSource"));
provAttrs.add(new Attribute("description","DB2 JDBC2-compliant
         + XA Driver"));

//create the provider
ObjectName jdbcProv = configService.createConfigData(session,node1,
      "JDBCProvider", "JDBCProvider",provAttrs);
// now plug in the classpath
configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);


// Search for RRA so we can link it to the data source
ObjectName rra = null;
ObjectName j2cra = ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
matches = configService.queryConfigObjects(session, node1, j2cra, null);
for (int i = 0; i < matches.length; i++) {
  if ( matches[i].getKeyProperty("_Websphere_Config_Data_Display_Name").equals("WebSphere Relational
    Resource Adapter") ) {
    rra = matches[i];
    break;
  }
}

// Prepare the attribute list
AttributeList dsAttrs = new AttributeList();
dsAttrs.add(new Attribute("name", dsName));
dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
dsAttrs.add(new Attribute("datasourceHelperClassname",
     "com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// this is where we make the link to "builtin_rra"
dsAttrs.add(new Attribute("description", "JDBC data source for
     + mark section CMP 2.0 test"));
dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

// Create the data source
System.out.println("  **  Creating data source");
ObjectName dataSource =
    configService.createConfigData(session,jdbcProv,"DataSource",
       "DataSource",dsAttrs);

// Add a propertySet.
AttributeList propSetAttrs = new AttributeList();
ObjectName resourcePropertySet =
  configService.createConfigData(session,dataSource,"propertySet",
       "",propSetAttrs);

// Add resourceProperty databaseName
AttributeList propAttrs1 = new AttributeList();
propAttrs1.add(new Attribute("name", "databaseName"));
propAttrs1.add(new Attribute("type", "java.lang.String"));
propAttrs1.add(new Attribute("value", dbName));

configService.addElement(session,resourcePropertySet,
     "resourceProperties",propAttrs1,-1);


// Create the corresponding J2CResourceAdapter Connection Factory object.
ObjectName jra = ConfigServiceHelper.createObjectName(null,
    "J2CResourceAdapter",null);

// Get all the J2CResourceAdapter, and add the data source
System.out.println("  **  Get all J2CResourceAdapters");
ObjectName[] jras = configService.queryConfigObjects(session, node1,
      jra, null);

int i=0;

for (;i< jras.length;i++) {
   System.out.println(ConfigServiceHelper.getConfigDataType(jras[i])
           + " " + i + " = "
           + jras[i].getKeyProperty(SystemAttributes.
           + _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
           + "\nFrom scope ="
           + jras[i].getKeyProperty(SystemAttributes.
           + _WEBSPHERE_CONFIG_DATA_ID));
   // quit on the first builtin_rra
   if (jras[i].getKeyProperty(SystemAttributes.
                     + _WEBSPHERE_CONFIG_DATA_DISPLAY_NAME)
       .equals("WebSphere Relational Resource Adapter")) {
      break;
   }
}


if (i >= jras.length) {
   System.out.println("Did not find builtin_rra; the J2CResourceAdapter object is
```

```
            + creating connection factory anyway" );
         break;
      } else {
         System.out.println("Found builtin_rra J2CResourceAdapter object at index
            + " + i + " creating connection factory" );
      }

      // Prepare the attribute list
      AttributeList cfAttrs = new AttributeList();
      cfAttrs.add(new Attribute("name", dsName + "_CF"));
      cfAttrs.add(new Attribute("authMechanismPreference","BASIC_PASSWORD"));
      cfAttrs.add(new Attribute("authDataAlias",authDataAlias));
      cfAttrs.add(new Attribute("cmpDatasource", dataSource ));
      // Make the link to data source's xmi:id
      ObjectName cf = configService.createConfigData(session,jras[i],
         "CMPConnectorFactory", "CMPConnectorFactory",cfAttrs);


      // ===== start Security section
      System.out.println("Creating an authorization data alias " + authDataAlias);

      // Find the parent security object
      ObjectName security = ConfigServiceHelper.createObjectName(null,
          "Security", null);
      ObjectName[] securityName = configService.queryConfigObjects(session,
          null, security, null);
      security=securityName[0];

      // Prepare the attribute list
      AttributeList authDataAttrs = new AttributeList();
      authDataAttrs.add(new Attribute("alias", authDataAlias));
      authDataAttrs.add(new Attribute("userId", uid));
      authDataAttrs.add(new Attribute("password", pw));
      authDataAttrs.add(new Attribute("description","Auto created alias for data source"));

      //create it
      ObjectName authDataEntry = configService.createConfigData(session,security,"authDataEntries",
     "JAASAuthData",authDataAttrs);
      // ===== end Security section

      // Save the session
      System.out.println("Saving session" );
      configService.save(session, false);

      // reload resources.xml
      reload(adminClient,true);
   } catch (Exception ex) {
      ex.printStackTrace(System.out);
      throw ex;
   }
}

/**
 * Get the DataSourceConfigHelperMbean and call reload() on it
 *
 * @param adminClient
 * @param verbose true - print messages to stdout
 */
public void reload(AdminClient adminClient,boolean verbose) {
   if (verbose) {
      System.out.println("Finding the Mbean to call reload()");
   }

   // First get the  Mbean
   ObjectName handle = null;
   try {
      ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
      Set s = adminClient.queryNames(queryName, null);
      Iterator iter = s.iterator();
      if (iter.hasNext()) handle = (ObjectName)iter.next();
   } catch (MalformedObjectNameException mone) {
      System.out.println("Check the program variable queryName" + mone);
   } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
      System.out.println("Cannot connect to the application server" + ce);
   }

   if (verbose) {
      System.out.println("Calling reload()");
   }
   Object result = null;
   try {
      result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
   } catch (MBeanException mbe) {
      if (verbose) {
         System.out.println("\tMbean Exception calling reload" + mbe);
      }
   } catch (InstanceNotFoundException infe) {
      System.out.println("Cannot find reload ");
   } catch (Exception ex) {
      System.out.println("Exception occurred calling reload()" + ex);
```

```
      }

      if (result==null && verbose) {
         System.out.println("OK reload()");
      }
   }
}
```

Example: Using the Java Management Extensions API to create a JDBC driver and data source for BMP beans, session beans, or servlets:

This code sample demonstrates how to configure a JDBC provider and data source, set an authorization alias for the data source, and reload the Mbean to make configuration changes.

```
// "This program may be used, run, copied, modified and distributed without royalty for the
// purpose of developing, using, marketing, or distributing."
//
// Product 5630-A36,  (C) COPYRIGHT International Business Machines Corp., 2001, 2002
// All Rights Reserved * Licensed Materials - Property of IBM
//
import java.util.*;
import javax.sql.*;
import javax.transaction.*;
import javax.management.*;

import com.ibm.websphere.management.*;
import com.ibm.websphere.management.configservice.*;
import com.ibm.ws.exception.WsException;


/**
 * Creates a node scoped resource.xml entry for a DB2 XA datasource.
 * The datasource created is for BMP use.
 *
 * Set the following:
 * set classpath=%classpath%;install_root/lib/bootstrap.jar;install_root/lib/j2ee.jar;
 * install_root/plugins/com.ibm.ws.bootstrap_6.1.0.jar;install_root/plugins/com.ibm.ws.emf_2.1.0.jar;
 * install_root/plugins/com.ibm.ws.runtime_6.1.0.jar;install_root/plugins/org.eclipse.emf.common_2.2.1.v200609210005.jar;
 * install_root/plugins/org.eclipse.emf.ecore_2.2.1.v200609210005.jar;install_root/runtime/com.ibm.ws.admin.client_6.1.0.jar;
 */
public class CreateDataSourceBMP {

   String dsName = "markSection"; // the display name of the datasource
                                  // this is also the JNDI name and the connection factory
   String dbName = "SECTION";     // database name
   String authDataAlias = "db2admin"; // an authentication data alias
   String uid = "db2admin";           // user ID
   String pw  = "db2admin";           //  password
   String dbclasspath = "D:/SQLLIB/java/db2java.zip"; // path to the database driver




   /**
    * Main method.
    */
   public static void main(String[] args) {
      CreateDataSourceBMP cds = new CreateDataSourceBMP();

      try {
         cds.run(args);
      } catch (com.ibm.ws.exception.WsException ex) {
         System.out.println("Caught this " + ex   );
         ex.printStackTrace();
         //ex.getCause().printStackTrace();
      } catch (Exception ex) {
         System.out.println("Caught this " + ex );
         ex.printStackTrace();
      }
   }


   /**
    * This method creates the datasource using JMX.
    *
    * The datasource created here is only written into resources.xml.
    * It is not bound into namespace until the server is restarted or an
    * application is started
    */
   public void run(String[] args) throws Exception {

      try {
         // Initialize the AdminClient.
         // Specify the SOAP_CONNECTOR_ADDRESS port of your server configuration.
         Properties adminProps = new Properties();
         adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
         adminProps.setProperty(AdminClient.CONNECTOR_HOST, "localhost");
```

```
     adminProps.setProperty(AdminClient.CONNECTOR_PORT, "8880");
     AdminClient adminClient = AdminClientFactory.createAdminClient(adminProps);

     // Get the ConfigService implementation.
     com.ibm.websphere.management.configservice.ConfigServiceProxy configService =
     new com.ibm.websphere.management.configservice.ConfigServiceProxy(adminClient);

     Session session = new Session();

     // Use this group to add to the node-scoped resource.xml.
     ObjectName node1 = ConfigServiceHelper.createObjectName(null, "Node", null);
     ObjectName[] matches = configService.queryConfigObjects(session, null, node1, null);
     node1 = matches[0];     // use the first node found

     // Use this group to add to the server1 scoped resource.xml.
     ObjectName server1 = ConfigServiceHelper.createObjectName(null, "Server", "server1");
     matches = configService.queryConfigObjects(session, null, server1, null);
     server1 = matches[0];   // use the first server found

     // Create the JDBCProvider
     String providerName = "DB2 JDBC Provider (XA)";
     System.out.println("Creating JDBCProvider " + providerName );

     // Prepare the attribute list
     AttributeList provAttrs = new AttributeList();
     provAttrs.add(new Attribute("name", providerName));
     provAttrs.add(new Attribute("implementationClassName", "com.ibm.db2.jdbc.DB2XADataSource"));
     provAttrs.add(new Attribute("description","DB2 JDBC2-compliant XA Driver"));

     //create it
     ObjectName jdbcProv = configService.createConfigData(session,node1,"JDBCProvider",
"JDBCProvider", provAttrs);
     // now add the classpath
     configService.addElement(session,jdbcProv,"classpath",dbclasspath,-1);



     // Search for RRA so we can link it to the datasource
     ObjectName rra = null;
     ObjectName j2cra = ConfigServiceHelper.createObjectName(null, "J2CResourceAdapter", null);
     matches = configService.queryConfigObjects(session, node1, j2cra, null);
     for (int i = 0; i < matches.length; i++) {
          if ( matches[i].getKeyProperty("_Websphere_Config_Data_Display_Name").equals("WebSphere Relational
            Resource Adapter") ) {
            rra = matches[i];
            break;
          }
        }


     // Prepare the attribute list
     AttributeList dsAttrs = new AttributeList();
     dsAttrs.add(new Attribute("name", dsName));
     dsAttrs.add(new Attribute("jndiName", "jdbc/" + dsName));
     dsAttrs.add(new Attribute("datasourceHelperClassname","com.ibm.websphere.rsadapter.DB2DataStoreHelper"));
     dsAttrs.add(new Attribute("statementCacheSize", new Integer(10)));
     dsAttrs.add(new Attribute("relationalResourceAdapter", rra));
// Make the link to "builtin_rra"
     dsAttrs.add(new Attribute("description", "JDBC Datasource for mark section CMP 2.0 test"));
     dsAttrs.add(new Attribute("authDataAlias",authDataAlias));

     // Create the data source
     System.out.println("  **  Creating datasource");
     ObjectName dataSource = configService.createConfigData(session,jdbcProv,"DataSource",
   "resources.jdbc:DataSource",dsAttrs);

     // Add a propertySet.
     AttributeList propSetAttrs = new AttributeList();
     ObjectName resourcePropertySet =configService.createConfigData(session,dataSource,
   "propertySet","",propSetAttrs);

     // Add resourceProperty databaseName
     AttributeList propAttrs1 = new AttributeList();
     propAttrs1.add(new Attribute("name", "databaseName"));
     propAttrs1.add(new Attribute("type", "java.lang.String"));
     propAttrs1.add(new Attribute("value", dbName));

     configService.addElement(session,resourcePropertySet,"resourceProperties",propAttrs1,-1);


     // ===== start Security section
     System.out.println("Creating an authorization data alias " + authDataAlias);

     // Find the parent security object
     ObjectName security = ConfigServiceHelper.createObjectName(null, "Security", null);
     ObjectName[] securityName = configService.queryConfigObjects(session, null, security, null);
     security=securityName[0];

     // Prepare the attribute list
     AttributeList authDataAttrs = new AttributeList();
```

```
        authDataAttrs.add(new Attribute("alias", authDataAlias));
        authDataAttrs.add(new Attribute("userId", uid));
        authDataAttrs.add(new Attribute("password", pw));
        authDataAttrs.add(new Attribute("description","Auto created alias for datasource"));

        //create it
        ObjectName authDataEntry = configService.createConfigData(session,security,"authDataEntries",
      "JAASAuthData",authDataAttrs);
        // ===== end Security section

        // Save the session
        System.out.println("Saving session" );
        configService.save(session, false);

        // reload resources.xml
        reload(adminClient,true);

      } catch (Exception ex) {
        ex.printStackTrace(System.out);
        throw ex;
      }
   }

   /**
    * Get the DataSourceConfigHelperMbean and call reload() on it
    *
    * @param adminClient
    * @param verbose true - print messages to stdout
    */
   public void reload(AdminClient adminClient,boolean verbose) {
      if (verbose) {
        System.out.println("Finding the Mbean to call reload()");
      }

      // First get the  Mbean
      ObjectName handle = null;
      try {
        ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
        Set s = adminClient.queryNames(queryName, null);
        Iterator iter = s.iterator();
        if (iter.hasNext()) handle = (ObjectName)iter.next();
      } catch (MalformedObjectNameException mone) {
        System.out.println("Check the program variable queryName" + mone);
      } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
        System.out.println("Cannot connect to the application server" + ce);
      }

      if (verbose) {
        System.out.println("Calling reload()");
      }
      Object result = null;
      try {
        result = adminClient.invoke(handle, "reload", new Object[] {}, new String[] {});
      } catch (MBeanException mbe) {
        if (verbose) {
          System.out.println("\tMbean Exception calling reload" + mbe);
        }
      } catch (InstanceNotFoundException infe) {
        System.out.println("Cannot find reload ");
      } catch (Exception ex) {
        System.out.println("Exception occurred calling reload()"  + ex);
      }

      if (result==null && verbose) {
        System.out.println("OK reload()");
      }
   }
}
```

Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool:

The following sample code is a JACL (WSadmin - scripting tool) script used to create a data source.

Use this script to create only data sources for which the product does *not* provide a template. For every JDBC provider WebSphere Application Server supports, the product provides a corresponding data source template. See the topic Creating configuration objects using the wsadmin tool for instructions on how to use the `createUsingTemplate` command to establish these data sources. For a complete list of supported JDBC providers (and therefore a complete list of data sources that must be created using a template), refer to the topic “Data source minimum required settings, by vendor” on page 1433.

This script sets up the following sample JDBC objects:

- Creates a data source *fvtDS_1*
- Creates a 4.0 data source *fvtDS_3*
- Creates a container-managed persistence (CMP) data source linked to *fvtDS_1*

Note: If you later modify the class path or native library path of the JDBC provider associated with your data source, you must restart every application server within the scope of that JDBC provider for the new configuration to work. Otherwise, you receive a data source failure message.

```
#AWE --  Set up XA DB2 data sources, both Version 4.0 and  Connector architecture (JCA)-compliant data sources


#UPDATE THESE VALUES:
#The classpath that will be used by your database driver
set driverClassPath "c:/sqllib/java/db2java.zip"

set server "server1"

set fvtbase "c:/wssb/fvtbase"

#Users and passwords..
set defaultUser1 "dbuser1"
set defaultPassword1 "dbpwd1"
set aliasName "alias1"

set databaseName1 "jtest1"
set databaseName2 "jtest2"
#END OF UPDATES

puts "Add an alias alias1"
set cell [$AdminControl getCell]
set sec [$AdminConfig getid /Cell:$cell/Security:/]

#-------------------------------------------------------
# Create a JAASAuthData object for component-managed authentication
#-------------------------------------------------------
puts "create JAASAuthData object for alias1"

set alias_attr    [list alias $aliasName]
set desc_attr     [list description "Alias 1"]
set userid_attr   [list userId $defaultUser1]
set password_attr [list password $defaultPassword1]
set attrs [list $alias_attr $desc_attr $userid_attr $password_attr]

set authdata [$AdminConfig  create JAASAuthData $sec $attrs]
$AdminConfig save

puts "Installing DB2 datasource for XA"

puts "Finding the old JDBCProvider.."
#Remove the old jdbc provider...
set jps [$AdminConfig list JDBCProvider]
foreach jp $jps {
 set jpname [lindex [lindex [$AdminConfig show $jp {name}] 0] 1]
 if {($jpname == "FVTProvider")} {
  puts "Removing old JDBC Provider"
  $AdminConfig remove $jp
  $AdminConfig save
 }
}


#Get the server name...
puts "Finding the server $server"
set servlist [$AdminConfig list Server]
set servsize [llength $servlist]
foreach srvr $servlist {
```

```
 set sname [lindex [lindex [$AdminConfig show $srvr {name}] 0] 1]
 if {($sname == $server)} {
                puts "Found server $srvr"
  set serv $srvr
 }
}

set desiredNodeName "myNode"
puts "Finding the Node"
set nodelist [$AdminConfig list Node]
foreach node $nodelist {
  set nodename [lindex [lindex [$AdminConfig show $node {name}] 0] 1]
  if {($nodename == $desiredNodeName)} {
    puts "node = $node"
    break
  }
}

puts "Finding the Resource Adapter"
set ralist [$AdminConfig list J2CResourceAdapter $node]
set ralistlen [llength $ralist]
foreach ra $ralist {
  set raname [lindex [lindex [$AdminConfig show $ra {name}] 0] 1]
  if {($raname == "WebSphere Relational Resource Adapter")} {
    set rsadapter $ra
  }
}
#Create an iSeries JDBC Provider for the data sources
puts "Creating the provider for com.ibm.db2.jdbc.app.UDBXADataSource"
set attrs1 [subst {{classpath $driverClassPath}
 {implementationClassName com.ibm.db2.jdbc.app.UDBXADataSource}{name "FVTProvider2"}
 {description "DB2 UDB for iSeries JDBC Provider"} {xa "true"}}]
set provider1 [$AdminConfig create JDBCProvider $serv $attrs1]
#Create the first data source
puts "Creating the datasource fvtDS_1"
set attrs2 [subst {{name fvtDS_1} {description "FVT DataSource 1"}}]
set ds1 [$AdminConfig create DataSource $provider1  $attrs2]

#Set the properties for the data source.
set propSet1 [$AdminConfig create J2EEResourcePropertySet $ds1 {}]

set attrs3 [subst {{name databaseName} {type java.lang.String} {value $databaseName1}}]
$AdminConfig create J2EEResourceProperty $propSet1 $attrs3

set attrs10 [subst {{jndiName jdbc/fvtDS_1} {statementCacheSize 10}
 {datasourceHelperClassname com.ibm.websphere.rsadapter.DB2DataStoreHelper}
 {relationalResourceAdapter {$rsadapter}} {authMechanismPreference "BASIC_PASSWORD"}
 {authDataAlias  $aliasName}}]
$AdminConfig modify $ds1 $attrs10


#Create the connection pool object
$AdminConfig create ConnectionPool $ds1 {{connectionTimeout 1000}
 {maxConnections 30} {minConnections 1} {agedTimeout 1000}
 {reapTime 2000} {unusedTimeout 3000} }


#Create the 4.0 data sources
puts "Creating the 4.0 datasource fvtDS_3"
set ds3 [$AdminConfig create WAS40DataSource $provider1 {{name fvtDS_3} {description "FVT 4.0 DataSource"}}]

#Set the properties on the data source
set propSet3 [$AdminConfig create J2EEResourcePropertySet $ds3 {}]

#These attributes should be the same as fvtDS_1
set attrs4 [subst {{name user} {type java.lang.String} {value $defaultUser1}}]
```

```
set attrs5 [subst {{name password} {type java.lang.String} {value $defaultPassword1}}]
$AdminConfig create J2EEResourceProperty $propSet3 $attrs3
$AdminConfig create J2EEResourceProperty $propSet3 $attrs4
$AdminConfig create J2EEResourceProperty $propSet3 $attrs5
set attrs10 [subst {{jndiName jdbc/fvtDS_3} {databaseName $databaseName1}}]
$AdminConfig modify $ds3 $attrs10

$AdminConfig create WAS40ConnectionPool $ds3 {{orphanTimeout 3000} {connectionTimeout 1000}
 {minimumPoolSize 1} {maximumPoolSize 10} {idleTimeout 2000}}

#Add a CMP connection factory for the JCA-compliant data source. This step is not necessary for
#Version 4 data sources, as they contain built-in CMP connection factories.
puts "Creating the CMP Connector Factory for fvtDS_1"
set attrs12 [subst {{name "FVT DS 1_CF"} {authMechanismPreference BASIC_PASSWORD}
 {cmpDatasource $ds1} {authDataAlias $aliasName}}]
set cf1 [$AdminConfig create CMPConnectorFactory $rsadapter $attrs12]

#Set the properties for the data source.
$AdminConfig create MappingModule $cf1 {{mappingConfigAlias "DefaultPrincipalMapping"} {authDataAlias "alias1"}}

$AdminConfig save
```

Using the DB2 Universal JDBC Driver to access DB2 for z/OS

The z/OS operating system requires that you configure the DB2 Universal JDBC Driver and your database to ensure interoperability. Within WebSphere Application Server, configure a JDBC provider object and a data source object to implement the driver capabilities for your applications.

Before you begin

Use only the following versions of the DB2 Universal JDBC Driver to connect with DB2 on z/OS; consult DB2 service updates for available enhancements on the version that you choose.

- The DB2 Universal JDBC Driver in DB2 UDB for z/OS Version 8. This version supports both driver Types 2 and 4.
- The DB2 Universal JDBC Driver for DB2 UDB for OS/390® and z/OS Version 7, as documented in APAR PQ80841. This version supports both driver Types 2 and 4.
- The DB2 Universal JDBC Driver with the feature *z/OS Application Connectivity to DB2 for z/OS*, which provides Type 4 connectivity only. If you install this version of the driver, you must configure a DB2 Universal JDBC Driver provider (XA) to access remote DB2 databases.

Note: If you are replacing the DB2 for 390 and z/OS Legacy JDBC driver with the DB2 Universal JDBC Driver, you can migrate your existing JDBC provider settings. Consult the topic ″Migrating from the JDBC/SQLJ Driver for OS/390 and z/OS to the DB2 Universal JDBC Driver″ in the Information Management Software for z/OS Solutions Information Center, which is located at http://publib.boulder.ibm.com/infocenter/dzichelp.

1. Install the driver class files and any necessary native files in an available HFS directory. (Native files are class files that some versions of the DB2 Universal JDBC Driver require for running on the z/OS operating system.)
2. Configure the driver and database for interoperability
   a. Bind the required DB2 packages

      As with any application that executes SQL statements in DB2 for z/OS, the Universal JDBC driver must first bind with DB2 the packages that represent the SQL statements to be executed.

      The specific details of the bind utility and bind process are described by the README provided with the installed DB2 Universal JDBC Driver. Refer to this README for details on how to setup and perform the required binding.

      Also note that the utility requires the server name (or IP address), the port number, and the database name (the database location on z/OS) for the target DB2. To get this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 system. This displays the IPADDR (IP address),

the SQL DOMAIN (server name), the TCPPORT number, and the LOCATION (database name/location) for you to use as input to the utility.

You must perform the bind process for each target DB2 that will be accessed using the DB2 Universal JDBC Driver.

b. Set up to handle in-doubt transactions

You must perform this setup once for each target DB2 for z/OS Version 7 location that is accessed using the DB2 Universal JDBC Driver Type 4 XA support.

Because DB2 for z/OS Version 7 does not implement Java Platform, Enterprise Edition (Java EE) XA support, the Type 4 driver XA processing uses DB2 V7 two-phase commit protocol and a table in each location (database) to store a list of global transactions that are in doubt (finished but not committed).

This table must be set up at each DB2 V7 location that is accessed. To do this, use the In-Doubt Utility, which is included as part of the installed DB2 Universal JDBC Driver. Use this utility to create the SYSIBM.INDOUBT Table that stores information about In-Doubt Global Transactions. This utility also binds the package T4XAIndbtPkg, which contains the SQL statements to insert and delete from the SYSIBM.INDOUBT Table. The T4XAIndbtPkg package is written with SQLJ.

This installation process requires that the target DB2 subsystem be configured with DDF enabled for incoming TCP/IP connections.

1) To enable DDF on the target DB2, issue the DB2 **-START DDF** command on that system.
2) This utility requires the server name (or IP address) and the port number for the target DB2 V7. To obtain this information, issue a DB2 **-DISPLAY DDF** command on the target DB2 V7 system. This displays the IPADDR (IP address), the SQL DOMAIN (server name), and the TCPPORT number that can be used as input to the utility.

To find more detailed information about the In-Doubt utility, refer to the *DB2® Universal Database™ for z/OS Version 7 Application Programming Guide and Reference for Java™* publication. (You can download it from the Library section of the DB2 Universal Database for z/OS Version 7 product information Web pages.) Within this publication, search for discussion about the utility under **`DB2T4XAIndoubtUtil`**, which is the official name of the In-Doubt utility.

Note: The previously described setup for in-doubt transactions is ***not*** a requirement for DB2 FOR z/OS Version 8 servers because DB2 FOR z/OS Version 8 natively supports XA commands over DRDA® and manages the In-Doubt Global Transactions internally.

c. Define a db2.jcc.propertiesFile

A db2.jcc.propertiesFile for use by DB2 Universal JDBC Driver Type 2 processing under WebSphere Application Server for z/OS can be created and specified as input to the driver. This runtime properties file is for use in specifying various runtime options that the DB2 Universal JDBC Driver uses for Type 2 connectivity. These options are specified as properties in the form of parameter=value. Refer to the README file packaged with the installed DB2 Universal JDBC Driver for a detailed description of each of the properties.

This file is not required; however, if it is not provided, universal driver default processing is performed.

Of specific interest is the db2.jcc.ssid property. This property specifies the DB2 subsystem identifier (not location name), to be used by the DB2 Universal JDBC Driver Type 2 processing as the local subsystem name to which it should connect. If this property is not provided, the driver uses the subsystem identifier that it finds in the DSNHDECP load module. If the installation wants to use the DSNHDECP load module to specify the subsystem identifier, this load module must be included in a steplib dataset in the servant region PROCs associated with each server that will use the DB2 identified by the subsystem ID. Refer to the README file packaged with the universal driver for more information on using this load module. If that DSNHDECP load module does not accurately reflect the desired subsystem, or if multiple subsystems might be using a generic DSNHDECP, the db2.jcc.ssid property must be specified.

Although the db2.jcc.propertiesFile is not required, if you choose to define the file, you must specify the fully qualified-hfs-filename. To do this, specify the file as a JVM System property as follows:

- **db2.jcc.propertiesFile = <fully-qualified-hfs-filename>**

Because the driver-general properties are typically specific to a driver load (for example, server) rather than all servers using the JDBC provider, it is best to set this JVM property at the server level. To define the **db2.jcc.propertiesFile=** property to the server level using the WebSphere Application Server for z/OS Administrative Console:

1) Under the WebSphere Application Server for z/OS Administrative Console, go to **Servers** > **Application Servers**, then click the server to which you want to add the JVM property.
2) On the selected server page, expand **Java and Process Management** and click **Process Definition** > **Servant**.
3) On the Servant page, click **Additional Properties**, then click **Java Virtual Machine**.
4) On the Java Virtual Machine page, click **Additional Properties**, then click **Custom Properties**.
5) On the Custom Properties page, scroll down and click **New** to configure a new JVM property for the selected server. The name of the property is **db2.jcc.propertiesFile**. The value of the property is the fully-qualified-hfs-filename that you created and initialized with the DB2 Universal JDBC Driver properties. These are the properties that you want the Type 2 driver to use for the selected server
6) Click **Ok**.
7) Click **Save** to save the new JVM property.

3. Define a JDBC provider for the DB2 Universal JDBC Driver. The JDBC provider object encapsulates the driver classes for implementation in WebSphere Application Server.
   a. From the WebSphere Application Server for z/OS Administrative Console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the *scope* at which applications can use the JDBC provider. (This scope becomes the scope of any data source that you associate with this provider.) You can choose a cell, node, cluster, or server. For more information, see Administrative console scope settings.
   c. Click **New**. This action causes the **Create a new JDBC Provider** wizard to launch.
   d. Use the first drop-down list to select DB2 for z/OS as your database type.
   e. Select the DB2 Universal JDBC Driver provider as your JDBC provider type in the second drop-down list.
   f. From the third drop-down list, select the implementation type that is necessary for your application.

      If your application does not require that connections support two-phase commit transactions, and you plan to use type 4 connectivity, choose **Connection Pool Data Source**. If you use the connection pool data source with type 2 connectivity, however, Application Server on z/OS uses RRS to process *both* one-phase and two-phase transactions.

      Note: Do not select **Connection Pool Data Source** if your installation has the z/OS Application Connectivity to DB2 for z/OS feature defined to WebSphere Application Server for z/OS. Only the XA implementation of the DB2 Universal JDBC Driver supports this feature.

      Choose **XA Data Source** if you plan to use driver type 4, and your application requires connections that support two-phase commit transactions. Use only driverType 4 connectivity for the XA data source.

      After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.
   g. Click **Next** after you have defined your database type, provider type, and implementation type. Now you see the wizard page Enter database class path information.

      Typically you do not need to change the class path that already populates the field. (That class path is the value of the WebSphere environment variable that is displayed on this page, in the form

of ${*DATABASE*_JDBC_DRIVER_PATH}.) Most likely you also do not need to change the native library path or the data source implementation class name.

h. Click **Next** to see a summary of your JDBC provider settings.

i. Click **Finish** if you are satisfied with the entire JDBC provider configuration. You now see the JDBC provider collection page, which displays your new JDBC provider in a table along with other providers that are configured for the same scope.

4. Define a data source. WebSphere Application Server uses the data source object to obtain database connections and manage those connections.

   a. From the WebSphere Application Server for z/OS Administrative Console, access the page for the data source version that your applications require. If you need support for two-phase transactions, use only a data source of the latest standard version. Version 4 data sources do not support connections that participate in two-phase transactions.

      You can reach the appropriate page in one of two ways:

      - Click **Resources** > **JDBC** > **Data sources**, or **Data sources (WebSphere Application Server Version 4)**.
      - Click **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**, or **Data sources (WebSphere Application Server Version 4)**.

   b. Select the *scope* at which applications can use the data source. You can choose a cell, node, cluster, or server. For more information, see the Administrative console scope settings article.

      Note: From this point onward, the steps for creating Data sources (WebSphere Application Server Version 4) differ from the steps for creating data sources of the latest standard version. To configure a Version 4 data source:

      - Click **New** to proceed to the console page for defining required properties.
      - On this properties page specify values for the fields that are grouped under the heading **Configuration**. Application Server requires these properties to implement your JDBC driver classes; consult “Data source minimum required settings, by vendor” on page 1433 to learn about acceptable values.
      - Save your configuration by clicking **OK**. You are now finished with the primary data source configuration tasks.
      - Optional: Define additional properties that are supported by the DB2 Universal JDBC provider. Application Server calls them *custom properties*, and requires that you set them on the data source as well. Begin by clicking the Custom Properties link that is now displayed on the administrative console page. You can learn about optional data source properties in the *Application Programming Guide and Reference for Java* for your version of DB2 for z/OS.

   c. Click **New**. This action causes the **Create a data source** wizard to launch and display the Enter basic data source information page. The first field is the scope field, which is read-only. This field displays your previous scope selection.

   d. Type a data source name in the Data source name field. This name identifies the data source for administrative purposes only.

   e. Type a Java Naming and Directory Interface (JNDI) name in the JNDI name field. WebSphere Application Server uses the JNDI name to bind application resource references to this data source. For more information on JNDI, consult the “Naming” on page 2048 article.

   f. Configure a component-managed alias to secure your data source if you plan to implement driverType 4 connectivity with the DB2 Universal JDBC Driver. If you plan to use driverType 2 connectivity, you do not have to set an alias. In this case the connection manager uses a default authentication alias, which is the user identity of a thread when that thread delivers a getConnection request.

      A component-managed alias consists of an ID and password that are specified in an application for data source authentication. Therefore, the alias that you set on the data source must be identical to

the alias in the application code. For more information on Java 2 Connector (J2C) security, see the Managing Java 2 Connector Architecture authentication data entries article.

To set a component-managed alias, either select an existing alias or create a new one.

- Use the drop-down list to select an existing component-managed authentication alias.
- To create a new alias, click the **create a new one** link. This action closes the data source wizard and triggers the administrative console to display the J2C authentication data collection page. Click **New** to define a new alias. Click **OK** to save your settings and view the new alias on the J2C authentication data collection page. Restart the data source wizard by navigating back to the data source collection page, selecting the appropriate scope, and clicking **New**.

g. Click **Next** to see the wizard page Select JDBC provider.

h. Either select an existing JDBC provider, or create a new provider.

To select an existing JDBC provider:

1) Click **Select an existing JDBC provider**.
2) Select a JDBC driver from the drop-down list.
3) Click **Next**. You now see the page entitled Enter database specific properties for the data source.

To create a new JDBC provider:

1) Click **Create new JDBC provider**.
2) Click **Next** to see the Create JDBC provider page.
3) Use the first drop-down list to select DB2 for z/OS as your database type.
4) Select the DB2 Universal JDBC Driver provider as your JDBC provider type in the second drop-down list.
5) From the third drop-down list, select the implementation type that is necessary for your application.

   If your application does not require that connections support two-phase commit transactions, choose **Connection Pool Data Source**. Both driverType 2 and driverType 4 connectivity implementations of the DB2 Universal JDBC Driver support connection pool data sources.

   Note: Do not select this provider if your installation has the z/OS Application Connectivity to DB2 for z/OS feature defined to WebSphere Application Server for z/OS. Only the XA implementation of the DB2 Universal JDBC Driver supports this feature.

   Choose **XA Data Source** if your application requires connections that support two-phase commit transactions. Applications that use this data source configuration have the benefit of container-managed transaction recovery. Use only driverType 4 connectivity for the XA implementation.

   After you select an implementation type, the wizard fills the name and the description fields for your JDBC provider. You can type different values for these fields; they exist for administrative purposes only.
6) Click **Next** after you have defined your database type, provider type, and implementation type. Now you see the wizard page Enter database class path information.

   Typically you do not need to change the class path that already populates the field. (That class path is the value of the WebSphere environment variable that is displayed on this page, in the form of ${*DATABASE*_JDBC_DRIVER_PATH}.) Most likely you also do not need to change the native library path or the data source implementation class name.
7) Click **Next**. You now see the page entitled Enter database specific properties for the data source.

i. Click **Use this data source in container managed persistence (CMP)** if container managed persistence (CMP) enterprise beans must access this data source.

j. Specify all of the remaining properties, as they are required for implementation of the DB2 Universal JDBC Driver. These properties include:

- Database name, which is the location name of the target database used when establishing connections with this data source
- driverType, which is the JDBC connectivity type used by the data source
- Server name, which is the TCP/IP address or host name for the Distributed Relational Database Architecture™ (DRDA) server.

  This property is required only if driverType is set to 4. This property is not used if driverType is set to 2.
- Port number, which is the TCP/IP port number where the DRDA server resides.

  Provide a value for this property only if driverType is set to 4. Do not set this property if driverType is set to 2.

k. Click **Finish** to save the configuration and exit the wizard. You now see the Data source collection page, which displays your new configuration in a table along with other data sources that are configured for the same scope.

What to do next

You can override the default values for some data source properties. Click your new data source link in the table to view the general configuration page for required data source properties. You can also define additional properties that are supported by the DB2 Universal JDBC Driver. Application Server requires that you set them as custom properties on the data source. Learn about optional data source properties in the *Application Programming Guide and Reference for Java* for your version of DB2 for z/OS.

Extend DB2 data source definitions at the application level

Extend data source definitions, which consist of non-core or custom properties, for DB2 data sources to add a greater level of application flexibility when you are using the DB2 Universal JDBC driver or DB2 Using IBM JCC driver. Use this feature to configure a DB2 data source in the application server with a core set of data source properties, and defer to individual applications to define any custom or non-core properties, like currentSchema or clientApplicationInformation, that you want to be application specific. You can also use these extended definitions to override any non-core or custom properties that are already defined for the data source. In addition, this feature can reduce the number of physical connections that the application server uses by employing one connection pool between resources that connect to the same data source.

Before you begin

You must be using a DB2 data source, and the data source must be configured in the application server with one of the following JDBC providers:

- DB2 Universal JDBC driver (Version 3.1 or higher)
- DB2 Using IBM JCC driver (Version 3.1 or higher)

About this task

Associate non-core properties with different resource references for a data source, and configure your applications to take advantage of these resource references to extend or override any non-core properties for the data source. You can choose to define new non-core or custom properties, or override any non-core properties that are already defined for the data source.

Also, applications can share the same connection pool in the application server, even though each application might have its own unique set of data source properties. There might be only one data source that is defined in the application server, and therefore only one connection pool, but to individual applications it would appear that there is more than one data source that is defined. This can result in:

- Reduction in memory consumption by the application server. Data source definitions will correspond to one connection pool, so there are fewer objects in memory when compared with a data source definition that corresponds to its own connection pool.

- Reduction in the number of open connections to the data source, which can reduce the memory consumption by the data source. The application server can reduce the number of idle connections by providing one connection pool that corresponds to multiple data sources, providing a more efficient use of connections.
- Avoidance of two-phase commit (XA) transactions in certain DB2 scenarios, when your applications use the get/use/close connection pattern. The application server can share connections between different resource references that have same set of core properties, even if they have different non-core properties, within the same transaction. This behavior can avoid two-phase commit processing if the connection sharing leads the application server to use one and only one physical connection.

1. Update the data source definitions for an application that is already installed.
   a. Navigate to the panel to manage the resource references for the application.
      - For applications that do not use container-managed persistence, click **Applications** → **WebSphere enterprise applications** → ***application_name*** → **Resource references**.
      - For applications that use container-managed persistence, click **Applications** → **WebSphere enterprise applications** → ***application_name*** → **Provide default data source mapping for modules containing 2.x entity beans**. You cannot add extended properties for individual CMP beans.
      - You can to configure two resource reference files on the same data source. This allows you to extend the custom properties on the data source to include two different schema names (currentSQLId on z/OS or currentSchema name in the custom properties) that can be used to exploit the capabilities of the application server. See “Configure two resource reference files on the same data source” on page 1497 for more information.

        Note: For pureQuery, if this is an XA data source you must define a new custom property on the data source where *property_name* = `downgradeHoldCursorsUnderXa` and boolean value = `true`.
   b. Navigate to the panel for extended data source properties. Click **Extended properties...** in the **Target Resource JNDI Name** column. If the data source does not support extended data source properties, you will receive an error when you try to apply these changes.
   c. Add a data source definition that is specific to that target resource. Click **New**, and complete the required fields. You cannot modify the following data source properties, which must be the same for all applications that use this data source:
      - accountingInterval
      - dataSourceName
      - databaseName
      - kerberosServerPrincipal
      - loginTimeout
      - logWriter
      - password
      - pkList
      - planName
      - portNumber
      - readOnly
      - securityMechanism
      - serverName
      - user
   d. Optional: Configure two res-refs and the isolation level in the ibm-ejb-jar-ext.xml

2. Optional: To avoid two-phase commit transactions, employ the get/use/close connection pattern. The get/use/close pattern is when an application gets a connection from a data source or connection factory and completes the current work within a single method, and the application does not call another method until that work is complete.
   a. Ensure that your applications use the get/use/close connection pattern.
   b. Enable the application server to verify use of the get/use/close connection pattern in applications, which will lead to the avoidance of two-phase commit processing if the sharing leads to one and only one physical connection being used.
      1) Click **Resources** → **JDBC** → **Data sources** → ***data_source*** → **WebSphere Application Server data source properties**
      2) Select **Optimize for get/use/close connection pattern with heterogenous pooling**. You will receive error messages if you select this option and your applications do not use the get/use/close connection pattern.

Related concepts

"Data sources" on page 1319
Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

"JDBC providers" on page 1319
Installed applications use JDBC providers to interact with relational databases.

Related tasks

"Using the DB2 Universal JDBC Driver to access DB2 for z/OS" on page 1489
The z/OS operating system requires that you configure the DB2 Universal JDBC Driver and your database to ensure interoperability. Within WebSphere Application Server, configure a JDBC provider object and a data source object to implement the driver capabilities for your applications.

Related reference

"WebSphere Application Server data source properties" on page 1469
Use this page to set advanced data source properties in the application server. These properties activate and configure services that the application server applies to data sources to customize connections within an application server. These properties do not affect connections within the database.

Extended data source properties:

Use this page to set the extended data source properties for a DB2 database. You can use these properties to allow an application to extend the custom properties for a data source or override any non-core properties that already exist for that data source.

To access this administrative console page:

- For applications that do not use container-managed persistence click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Resource references**
- For applications that use container-managed persistence click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Provide default data source mapping for modules containing 2.x entity beans**.

For data sources that use the DB2 Universal JDBC driver or DB2 Using IBM JCC Driver, click **Extended properties...** in the **Target Resource JNDI Name** column.

Name: Specifies the name (or key) for the property.

Each property name must be unique. If the same name is used for multiple properties, the value specified for the first property that has that name is used.

Do not start your property names with `was`, because this prefix is reserved for properties that are predefined in the application server.

Data type String

Value:

Specifies the property value.

Data type Variable

Configure two resource reference files on the same data source:

You can configure two resource reference files on the same data source. This allows you to extend the custom properties for the data source to be extended to include two different schema names (currentSQLId on z/OS or currentSchema name in the custom properties) that can be used to exploit the capabilities of the application server.

About this task

When an EntityManager is created, the application server obtains a connection to the database. When you are using a pessimistic transaction, the EntityManager will retain that connection until the EntityManager is closed. When there are two EntityManagers that extend the data source definitions, the `openjpa.jdbc.TransactionIsolation` property might cause a problem with the transaction. This property can be found in the `persistence.xml` file in the following entry:

```
property name="openjpa.jdbc.TransactionIsolation" value="read-committed"
```

In order to satisfy this request, Java Persistence API (JPA) will obtain a connection and immediately call `setTransactionIsolation(READ_COMMITTED)`. When you have two EntityManagers share a single physical connection to the database, the first EntityManager creates a connection to the database and involves that connection in a transaction. When the second EntityManager creates a connection, it is not able to change the isolation level.

You can avoid this problem by creating two resource reference files in the same data source. You can create the resource references with Rational Application Developer or by editing the XML files. You will need to make changes to the `ejb-jar.xml`, `ibm-ejb-jar-bnd.xml`, `ibm-ejb-jar-ext.xml`, `persistence.xml` files.

Note: For pureQuery, if this is an XA data source you must define a new custom property on the data source where *property_name* = `downgradeHoldCursorsUnderXa` and boolean value = `true`.

See the following sections for information on how to accomplish this:

- Configure two resource reference files on the same data source using Rational Application Developer.
- Configure two resource reference files on the same data source by editing the XML files.
- Configure two resource reference files on the same data source using Rational Application Developer.
  1. Edit the `ejb-jar.xml` file
     a. Create the deployment descriptor if it doesn't already exist:
        1) Go to the context menu of the Enterprise Java beans (EJB) project and select **Java EE** → **Generate Deployment Descriptor Stub**.
     b. Edit the deployment descriptor:
        1) Go to the project's META-INF directory and select the `ejb-jar.xml` file.
     c. Add the enterprise bean's element to the deployment descriptor if it doesn't already exist:
        1) In the left hand pane, select **EJB Project node**. Click **Add**.

2) In the dialog, select **Enterprise Beans**. Click **OK**.

d. Add the session bean to the deployment descriptor if it's not already there:
   1) In the left pane, select **Enterprise Beans**. Click **Add**.
   2) Select **Session Bean**. Click **OK**.
   3) Enter the name of the session bean in the dialog. Click **OK**.
   4) In the right-side pane, enter the `EJB Class`, the business local, and the business remote interfaces.

e. Add the resource reference elements to the session bean:
   1) Select the session bean in the left pane and click **Add**.
   2) Select **Resource Reference**. Click **OK**.
   3) In the **Add Resource Reference** dialog, enter the name, type, authentication and sharing scope fields. Click **OK**.
   4) Repeat for the second resource reference.

f. Save the editor

2. Edit the `ibm-ejb-jar-bnd.xml` file
   a. Create the WebSphere EJB bindings descriptor if it doesn't already exist
      1) Go to the context menu of the Enterprise JavaBeans (EJB) project and select **Java EE** → **Generate WebSphere Bindings Deployment Descriptor**.
   b. Edit the bindings descriptor:
      1) In the project's META-INF directory, select the `ibm-ejb-jar-bnd.xml` file.
   c. Add a binding element for the session bean to the bindings descriptor:
      1) In the left pane, select **EJB Jar Bindings** node and click **Add**.
      2) In the dialog, select **Session** and click **OK**.
      3) In the right pane, enter the name of the session bean.
   d. Add the bindings for the resource references to the session bean:
      1) In the left pane, select the session bean and click **Add**.
      2) In the dialog, select **Resource Reference** and click **OK**.
      3) In the left pane, select the resource reference.
      4) In the right pane, enter the name and binding name for the reference
      5) Repeat for the second resource reference.
   e. Save the editor
3. Edit the `ibm-ejb-jar-ext.xml` file.
   a. Create the WebSphere EJB extensions descriptor if it doesn't already exist:
      1) Go to the context menu of the Enterprise JavaBeans (EJB) project and select **Java EE** → **Generate WebSphere Extensions Deployment Descriptor**.
   b. Edit the extensions descriptor:
      1) In the project's META-INF directory, select the `ibm-ejb-jar-ext.xml` file.
   c. Add an extensions element for the session bean to the extensions descriptor:
      1) In the left pane, select the **EJB Jar Extensions** node and click **Add**.
      2) In the dialog, select **Session** and click **OK**.
      3) In the right pane, enter the name of the session bean.
   d. Define the isolation level for the resource references:
      1) In the left pane, select the session bean and click **Add**.
      2) In the dialog, select **Resource Reference** and click **OK**.
      3) In the left pane, select the resource reference.
      4) In the right pane, enter the name and the isolation level.

5) Repeat for the second resource reference.

e. Save the editor

4. Edit the `persistence.xml` file.
   a. Add JPA support to the EJB project, which will create the persistence.xml file:
      1) From the project's context menu, select **Properties**.
      2) In the left pane, select the **Project Facets** node.
      3) In the right pane, check the box beside **Java Persistence**.
      4) Click **OK**.
   b. Edit the `persistence.xml` file:
      1) In the project's META-INF directory, select the `persistence.xml` file
   c. The created `persistence.xml` file already contains a persistence unit definition. Edit this `persistence.xml` file
      1) In the left pane, select the **Persistence Unit** node. Set the name, JTA data source and exclude the unlisted classes fields.
   d. Create a new persistence unit definition in the file:
      1) In the left pane, select the **Persistence** node and click **Add** .
      2) In the dialog, select **Persistence Unit** and click **OK**.
      3) In the left pan, select the **Persistence Unit** node. Set the name, JTA data source and exclude the unlisted classes fields.
   e. Add classes to the persistence unit:
      1) In the left pane, select the persistence unit node and click **Add** .
      2) In the dialog, select **Class** and click **OK**.
      3) In the right pane, enter the name of the class.
      4) Repeat for each class.
   f. Add properties to the persistence unit
      1) If there is not already a **Properties** element in the file, select the persistence unit node in the left pane and click **Add**.
      2) In the dialog, select **Properties** and click **OK**.
      3) In the left pane, select the **Properties** node and click **Add**.
      4) In the dialog, select **Property** and click **OK**.
      5) In the right pane, enter the name and value for the property
      6) Repeat the previous three steps to add the additional properties.
   g. Save the editor

- Configure two resource reference files on the same data source by editing the XML files.
  1. Edit the `ejb-jar.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar id="ejb-jar_ID" metadata-complete="false" version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd">
    <enterprise-beans>
        <session>
            <ejb-name>NewOrderSessionFacadeBean</ejb-name>
            <business-local>neworderssession.ejb3.NewOrderSessionFacade</business-local>
            <business-remote>neworderssession.ejb3.NewOrderSessionFacadeRemote</business-remote>
            <ejb-class>neworderssession.ejb3.NewOrderSessionFacadeBean</ejb-class>
            <session-type>Stateless</session-type>
   <resource-ref>
    <description></description>
    <res-ref-name>jdbc/ERWWDataSourceV5</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
   <resource-ref>
    <description></description>
    <res-ref-name>jdbc/ERWWDataSourceV5_HP</res-ref-name>
```

```
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
   </resource-ref>
       </session>
     </enterprise-beans>
</ejb-jar>
```

2. Edit the `ibm-ejb-jar-bnd.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar-bnd xmlns="http://websphere.ibm.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee http://websphere.ibm.com/xml/ns/javaee/ibm-ejb-jar-bnd_1_0.xsd"
 version="1.0">
 <session name="NewOrderSessionFacadeBean" simple-binding-name="ejb/session/NewOrderSessionFacadeBean">
  <resource-ref name="jdbc/ERWWDataSourceV5" binding-name="jdbc/ERWWDataSourceV5"></resource-ref>
  <resource-ref name="jdbc/ERWWDataSourceV5_HP" binding-name="jdbc/ERWWDataSourceV5"></resource-ref>
 </session>
</ejb-jar-bnd>
```

3. Edit the `ibm-ejb-jar-ext.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar-ext xmlns="http://websphere.ibm.com/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://websphere.ibm.com/xml/ns/javaee http://websphere.ibm.com/xml/ns/javaee/ibm-ejb-jar-ext_1_0.xsd"
 version="1.0" metadata-complete="true">
 <session name="NewOrderSessionFacadeBean">
  <resource-ref name="jdbc/ERWWDataSourceV5"
   isolation-level="TRANSACTION_READ_COMMITTED" />
  <resource-ref name="jdbc/ERWWDataSourceV5_HP"
   isolation-level="TRANSACTION_READ_COMMITTED" />
 </session>
</ejb-jar-ext>
```

4. Edit the `persistence.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 version="1.0" xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="NewOrderSessionEJB3">
        <jta-data-source>java:comp/env/jdbc/ERWWDataSourceV5</jta-data-source>
        <class>warehouseejb3.WarehouseJPA</class>
        <class>districtejb3.DistrictJPA</class>
        <class>customerejb3.CustomerJPA</class>
        <class>stockejb3.StockJPA</class>
        <class>orderejb3.OrderJPA</class>
        <class>orderlineejb3.OrderlineJPA</class>
        <class>neworderejb3.NewOrderJPA</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
  <property name="openjpa.LockManager" value="pessimistic"/>
  <property name="openjpa.ReadLockLevel" value="read"/>
  <property name="openjpa.WriteLockLevel" value="write"/>
  <property name="openjpa.LockTimeout" value="30000"/>
     <property name="openjpa.FetchBatchSize" value="1" />
     <property name="openjpa.jdbc.TransactionIsolation" value="read-committed" />
     <property name="openjpa.Log" value="none"/>
     </properties>
    </persistence-unit>
    <persistence-unit name="ItemEJB3">
        <jta-data-source>java:comp/env/jdbc/ERWWDataSourceV5_HP</jta-data-source>
        <class>itemejb3.ItemJPA</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
  <property name="openjpa.LockManager" value="pessimistic"/>
  <property name="openjpa.ReadLockLevel" value="read"/>
  <property name="openjpa.WriteLockLevel" value="write"/>
  <property name="openjpa.LockTimeout" value="30000"/>
     <property name="openjpa.FetchBatchSize" value="1" />
     <property name="openjpa.jdbc.TransactionIsolation" value="read-committed" />
     <property name="openjpa.Log" value="none"/>
     </properties>
    </persistence-unit>
</persistence>
```

Related tasks

"Extend DB2 data source definitions at the application level" on page 1494
Extend data source definitions, which consist of non-core or custom properties, for DB2 data sources to add a greater level of application flexibility when you are using the DB2 Universal JDBC driver or DB2 Using IBM JCC driver. Use this feature to configure a DB2 data source in the application server with a core set of data source properties, and defer to individual applications to define any custom or non-core properties, like currentSchema or clientApplicationInformation, that you want to be application specific. You can also use these extended definitions to override any non-core or custom properties that are already defined for the data source. In addition, this feature can reduce the number of physical connections that the application server uses by employing one connection pool between resources that connect to the same data source.

Enabling trusted context for DB2 databases

Enable trusted context in your applications to improve how the application server interacts with DB2 database servers. Use trusted connections to preserve the identity records of clients that are connecting to a DB2 database through your applications; trusted connections can provide a more secure environment by granting access based on the identity of those users.

Before you begin

Ensure that the following prerequisites are met before enabling trusted connections:

- You are using a database server that is running DB2 Version 9.5 or later for AIX, HP-UX, Linux, Solaris, or Windows operating systems or DB2 Version 9.1 or later for z/OS. See the list of list of supported software for the application server for more support information.
- You do not need to be connected to the database to configure trusted context in the application server.
- Trusted context is enabled for the DB2 database.
- Global security is enabled. Read about Setting up and enabling security for more information on configuring security.

About this task

Note: With trusted connections you can:

- Access the DB2 database with the caller identity, obviating the need to create a new connection for every user.
- Preserve the identity of the end-user when the application server is interacting with the database.
- Strengthen database security by avoiding granting all of the privileges to a single user.
- Improve performance, as compared to the existing model of using the resetConnection() method to take advantage of identity propagation.

Note: Non-trusted connections cannot be used as trusted connections. If the connection pool contains only non-trusted connections and a request comes in for a trusted connection, a new request will be sent to the database for the trusted connection.

Enable trusted context for your applications.

- Enable trusted context when you are installing a new application.
  1. Perform a typical installation for the application until you reach **Step 7: Map resource references to resources** in the installation wizard.
  2. In **Step 7: Map resource references to resources**, select **Use trusted connections (one-to-one mapping)** in the **Specify authentication method** section.
  3. Select an authentication alias from the list that matches an alias that is already defined in the DB2 data source. If you do not have an alias defined that is suitable, continue with the installation, and enable trusted context after the application is installed.

Note: You can specify a default user (UNAUTHENTICATED) to be used if no client identity is available, but that default ID (UNAUTHENTICATED) must also exist in the DB2 database. If the com.ibm.mapping.unauthenticatedUser is set to null or an empty string, then the application server will use the default user (UNAUTHENTICATED). Read about setting the com.ibm.mapping.unauthenticatedUser property for more information on this property.

4. Select a data source from the table that has trusted context enabled.
5. Click **Apply**.
6. Edit the properties of the custom login configuration. Read “Setting the security properties for trusted connections” on page 1503.
7. Finish the installation wizard.

- Enable trusted context on an application that is already installed.

Note: Remove the propagateClientIdentityUsingTrustedContext custom property for the DB2 data source, if it is present. If the propagateClientIdentityUsingTrustedContext is enabled, the application server will issue the following warning at run time:

```
IDENTITY_PROPAGATION_PROP_WARNING=DSRA7029W: The propagateClientIdentityUsingTrustedContext
 custom property for the Datasource is no longer used, value will be ignored.
```

The application server will determine at run time if the request is using trusted context, and the application server will enable trusted context based on that information. Therefore, the same data source in the application server can be used for both trusted and non-trusted access.

1. Click **Websphere enterprise applications** > *application_name*.
2. Click **Resource references** from the **Resources** heading.
3. Select **Use trusted connections (one-to-one mapping)** in the **Specify authentication method** section.
4. Select an authentication alias from the list that matches an alias that is already defined in the DB2 data source. If you do not have an alias defined that is suitable, define a new alias.
   a. Click **JDBC** > **Data sources** > *data_source_name*.
   b. Click **JAAS - J2C authentication data** from the **Related Items** heading.
   c. Click **New**.
   d. Define the properties for the alias in **General properties**.
   e. Click **OK**.

   Note: You can specify a default user (UNAUTHENTICATED) to be used if no client identity is available, but that default ID (UNAUTHENTICATED) must also exist in the DB2 database. If the com.ibm.mapping.unauthenticatedUser is set to null or an empty string, then the application server will use the default user (UNAUTHENTICATED). Read about setting the com.ibm.mapping.unauthenticatedUser property for more information on this property.
5. Select a data source from the table that has trusted context enabled.
6. Click **Apply**.
7. Edit the properties of the custom login configuration. Read “Setting the security properties for trusted connections” on page 1503.

What to do next

Be aware of the following error conditions that can occur if trusted context is not configured properly:

- The application server will issue a warning if you use the TrustedConnectionMapping login configuration and the database server does not support trusted context. The application server will then return a normal, non-trusted connection. If you are using a DB2 database for the database server, and it doesn't support trusted connections, then the DB2 database server will throw an exception.
- The application server will throw the following exception if you use the TrustedConnectionMapping login configuration and ThreadIdentity is specified:

```
IDENTITY_PROPAGATION_CONFLICT2_ERROR=DSRA7028E: You cannot use the TrustedConnectionMapping
 login configuration when the ThreadIdentity property is enabled.
```

- The application server will throw the following exception if you use the TrustedConnectionMapping login configuration and reauthentication is specified:

```
IDENTITY_PROPAGATION_CONFLICT1_ERROR=DSRA7025E: The reauthentication custom property for
 the Datasource cannot be enabled when you are using the TrustedConnectionMapping login configuration.
```

Related concepts

"Data sources" on page 1319
Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

"JDBC providers" on page 1319
Installed applications use JDBC providers to interact with relational databases.

Related tasks

"Using the DB2 Universal JDBC Driver to access DB2 for z/OS" on page 1489
The z/OS operating system requires that you configure the DB2 Universal JDBC Driver and your database to ensure interoperability. Within WebSphere Application Server, configure a JDBC provider object and a data source object to implement the driver capabilities for your applications.

Setting the security properties for trusted connections:

Trusted connections are a solution that can pass the requesting user identity to DB2 and also take full advantage of the connection pooling. Utilizing the DB2 trusted context object, the trusted connection is used to separate the identity used to establish the connection from the identity that accessed the DB2 server services. The connection is established by a user whose credentials are authorized by the DB2 server to open the connection and trusted by the DB2 server to assert the identity of the requesting users when accessing the DB2 server from the application.

Before you begin

To use the trusted connection functionality, you must be running at database server with DB2 Version 9.5 or later for AIX, HP-UX, Linux, Solaris, or Windows operating systems or DB2 Version 9.1 or later for z/OS. Trusted connections can be used if the application server is installed on iSeries systems, as long as a supported version of DB2 is installed on a platform other than iSeries systems, and the DB2 universal driver is used. See the list of list of supported software for the application server for more support information. An existing J2EE connector (J2C) data alias must exist for passing user credentials to the DB2 server when establishing a connection, meaning container authorization must be used.

Read about "Enabling trusted context for DB2 databases" on page 1501 for steps to configure the application server to use trusted connections.

About this task

Trusted connections support client identity propagation while taking advantage of connection pooling to reduce the performance penalty of closing and reopening connections with a different identity. When you select **Use trusted connection (one-to-one mapping)** for the connection mapping, five custom properties are created. Review these properties to ensure that the default values of these properties correspond with your intended settings.

1. Click **Enterprise applications** → ***application_name*** → **Resource references** → **Resources** panel in the administrative console.
2. Select the correct enterprise bean, and click **Mapping Properties** to view the properties that are set by default when you configured the trusted connection.
3. Confirm that the default values assigned to these properties are correct for your environment.

Table 11. Security Properties

Property	Default Value	Information
com.ibm.mapping.authDataAlias	none	The value that is assigned for this property is the value that you selected from the menu list.
com.ibm.mapping.propagateSecAttrs	false	A false value for this property specifies that the security attributes are not propagated. You can change this value to true to add the RunAs subject as an opaque token in the IdentityPrincipal object.
com.ibm.mapping.targetRealmName	null	If this value is not specified or null, the security run time process will use the current user realm name. This process assumes that the Enterprise Information System (EIS) is using the current user realm. In this context, a realm is a logical representation of the user repository. If the application server and DB2 server are using different user repositories, the value of this property should be set to the realm name of the DB2 server. This enables a principal or credential mapping to be set at the target EIS.
com.ibm.mapping.unauthenticatedUser	UNAUTHENTICATED	This property is a user identity that is used by the EIS to indicate a user identity that is unauthenticated. This is defined at com.ibm.ISecurityUtilityImpl.SecConstants.java public final static String UnauthenticatedString = ″UNAUTHENTICATED″
com.ibm.mapping.useCallerIdentityproperty	false	A false value for this property specifies the Run As identity is asserted in the IdentityPrincipal object. Change the value of this property to true if you want to assert the caller identity in the IdentityPrincipal object instead of the Run As identity.

4. Click **OK** to confirm all the current values.
5. Click **OK** and **Save** on the Resource references panel to save your changes to the master configuration.

Results

After the completion of these steps and a restart of the application server, trusted connections will be used with the chosen mapping properties to connect with the DB2 database server.

Trusted connections with DB2:

Trusted connections allow for the application server to use DB2 Trusted Context objects to establish connections with a user whose credentials are trusted by the DB2 server to open the connection. By establishing a Trusted Context, this user is then trusted to assert other user identities on the DB2 server without the expense of reauthentication. This also enhances the security of your DB2 database by eliminating the need to assign all privileges to a single user. Implementing trusted connections results in client identity propagation while leveraging connection pooling to eliminate the performance penalty of closing and reopening connections with a different identity.

Note: To use the trusted connection functionality you must be using DB2 Version 9.5 or later for AIX, HP-UX, Linux, Solaris, or Windows operating systems or DB2 Version 9.1 or later for z/OS. You can use trusted connections if version 7.0 is installed on an iSeries system as long as a supported version of DB2 is installed on a platform other than an iSeries system and the DB2 universal driver is being used. See the list of list of supported software for the application server for more support information.

To reduce the significant expense of establishing new connections, the connection manager maintains a connection pool in which each connection is tracked by the credential originally used to open the connection. When an application needs a connection, the connection manager uses the credential object to match a free connection from the connection pool. If no free connection is available and the maximum

number of connections has not been reached, the connection pool manager opens a new connection using that credential object. This connection mapping is the default connection mapping used by the application server and is known as a many-to-one credential mapping because the connection is opened using the credential object in the subject, which is usually not the same as the RunAs identity. This simple mapping supports easy connection pooling, but the caller identity is never propagated to database server.

To propagate the caller identify to the database server, you can plug in a Java Authentication and Authorization Service (JAAS) login module. Using this method, you would map the application server user credential to the user credential suitable for the database server security realm. This approach maintains the caller identity, but does not use connection pooling.

Trusted connections are used instead of the default mapping or a JAAS mapping to connect to the data source. Trusted connections support client identity propagation and can also use connection pooling to reduce the performance penalty of closing and reopening connections with a different identity. Trusted connections use the DB2 trusted context object.

The DB2 trusted context is an object that the database administrator defines and that contains a system authorization ID and a set of trust attributes. The trust attributes identify characteristics of a connection that are required for the connection to be considered trusted. The relationship between a database connection and a trusted context is established when the connection to the DB2 server is created. After a trusted context is defined, and an initial trusted connection to the DB2 database server is made, the application server can use that database connection from a different user without a full reauthentication. This is because an authentication token is required with the user identity. The database authenticates the user and then verifies the user authorization to access the database before allowing any database requests to be processed on behalf of that user.

Using the trusted connection provides the needed plug-in points to support adding your own secure implementation of the DB2 trusted context. Trusted connections separate the identity used to establish the connection from the identity that accesses the back-end server services. The connection is established by a user whose credentials are trusted by the DB2 server to open the connection. The same user is also then trusted to assert the identity of the other users. This assertion also helps strengthen database security by eliminating the need to grant all privileges to a single user.

When the application requests a connection to the database, the connection manager can find any idle trusted connection and assert the user identity to the backend server. All the operations performed on the backend server are from the asserted user identity. The use of an identity mapping may still be needed if the back-end server uses a different user repository than that of the application server.

A new mapping configuration called TrustedConnectionMapping implements trusted connections. The TrustedConnectionMapping configuration maps the RunAs subject to a resource subject that contains the following elements:

- A resource principal object that this resource subject represents
- A PasswordCredential object in the private credential set
- An IdentityPrincipal object in the principal set

The principal object represents the RunAs identity. The PasswordCredential object contains a user ID and password to be used by the resource adapter to establish the trusted connection. The IdentityPrincipal object by default contains the RunAs identity, but can be changed to use the identify of the caller. The IdentityPrincipal object also contains an original user identity that represents the user who sent the request initially, an optional realm name that indicates the set of registries where the user identity is defined and an optional security token, which is a serialized security context of the user.

Enabling trusted context with authentication:

Enable trusted context in your applications to improve how the application server interacts with DB2 database servers. Use trusted connections to preserve the identity records of clients that are connecting to

a DB2 database through your applications; trusted connections can provide a more secure environment by granting access based on the identity of those users. DB2 provides an option for trusted connections in which a password is required when switching the user identity. You can configure the application server to use trusted connections with authentication, and plug-in your own code to take advantage of trusted context with authentication.

Before you begin

Refer to the topic on enabling trusted context for DB2 databases to ensure that trusted connections are properly configured for the application server.

About this task

If the WITH AUTHENTICATION option is specified when the trusted context is created, the database requires that you provide an authentication token with the end user's identity. The database authenticates the end user and verifies the end user's authorization to access the database before the database allows any requests to be processed.

1. Set useTrustedContextWithAuthentication custom property to true for the DB2 data source.
   a. Click **JDBC** → **Data sources**.
   b. Click the name of the data source that you want to configure.
   c. Click **Custom properties** from the **Additional Properties** heading.
   d. Click **New**.
   e. Complete the required fields. Use the following information:

Table 12. Custom property panel

Name	Value
useTrustedContextWithAuthentication	true

   If the useTrustedContextWithAuthentication custom property is not set to true, the application server will provide an implementation of reusing DB2 trusted connections without authentication at run time. In this case you are not required to provide anything to use the trusted context feature.
2. Use the login configuration for TrustedConnectionMapping, as described in the topic on enabling trusted context for DB2 databases.
3. Extend the DataStoreHelper class, and provide the implementation for the getPasswordForUseWithTrustedContextWithAuthentication method as described in the topic on developing a custom DataStoreHelper class. At run time, the application server will call this method to return the password that the application server is required to use to switch the trusted context identity when you have enabled trusted context with authentication. The password that is returned by this method will be sent to the database when the application server switches trusted context identities, and the password will not be stored by the application server.

   This application server only calls this method if the following is true:
   - You set the useTrustedContextWithAuthentication data source custom property to true.
   - You use the TrustedConnectionMapping login configuration.

   The following is an example of the getPasswordForUseWithTrustedContextWithAuthentication method:

```
public String getPasswordForUseWithTrustedContextWithAuthentication(String identityname, String realm)
                throws SQLException
   {
      return customersOwnUtility().getPassword(identityname)   // customers use their own
                                                               // implementation to get the password
   }
```

   Note: You cannot enable the useTrustedContextWithAuthentication custom property for the data source without overwriting the getPasswordForUseWithTrustedContextWithAuthentication method in the DataStoreHelper class to get the password for switching the identity for trusted

connections. If you do not provide implementation for the getPasswordForUseWithTrustedContextWithAuthentication method, the application server will throw an exception with the following message at run time:

```
TRUSTED_WITH_AUTHENTICATION_IMPLEMENTATION_ERROR=DSRA7033E: You cannot enable the
useTrustedContextWithAuthentication custom property for the data source without
overwriting the getPasswordForUseWithTrustedContextWithAuthentication DataStoreHelper.
TRUSTED_WITH_AUTHENTICATION_IMPLEMENTATION_ERROR.explanation=The
useTrustedContextWithAuthentication custom property is enabled, but the implementation
code for the DataStoreHelper method that will return the password that the application
server will use to switch the identity is not provided.
TRUSTED_WITH_AUTHENTICATION_IMPLEMENTATION_ERROR.useraction=Overwrite the
getPasswordForUseWithTrustedContextWithAuthentication DataStoreHelper method and
provide the implementation code that will return the password, or set the
useTrustedContextWithAuthentication custom property for the data source to false.
```

Related concepts

"Trusted connections with DB2" on page 1504
Trusted connections allow for the application server to use DB2 Trusted Context objects to establish connections with a user whose credentials are trusted by the DB2 server to open the connection. By establishing a Trusted Context, this user is then trusted to assert other user identities on the DB2 server without the expense of reauthentication. This also enhances the security of your DB2 database by eliminating the need to assign all privileges to a single user. Implementing trusted connections results in client identity propagation while leveraging connection pooling to eliminate the performance penalty of closing and reopening connections with a different identity.

"Data sources" on page 1319
Installed applications use a *data source* to obtain connections to a relational database. A data source is analogous to the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection factory, which provides connectivity to other types of enterprise information systems (EIS).

"JDBC providers" on page 1319
Installed applications use JDBC providers to interact with relational databases.

Related tasks

"Developing a custom DataStoreHelper class" on page 1519
Apply the WebSphere extension, GenericDataStoreHelper class, to create your own data store helper for data sources that the application server does not support. With this helper class, your JDBC configuration can use database-specific functions during transactions.

"Setting the security properties for trusted connections" on page 1503
Trusted connections are a solution that can pass the requesting user identity to DB2 and also take full advantage of the connection pooling. Utilizing the DB2 trusted context object, the trusted connection is used to separate the identity used to establish the connection from the identity that accessed the DB2 server services. The connection is established by a user whose credentials are authorized by the DB2 server to open the connection and trusted by the DB2 server to assert the identity of the requesting users when accessing the DB2 server from the application.

"Using the DB2 Universal JDBC Driver to access DB2 for z/OS" on page 1489
The z/OS operating system requires that you configure the DB2 Universal JDBC Driver and your database to ensure interoperability. Within WebSphere Application Server, configure a JDBC provider object and a data source object to implement the driver capabilities for your applications.

Configuring the application server and DB2 to authenticate with Kerberos

Configure the application server to use Kerberos as the authentication mechanism for the application server that connects to a DB2 database server. The Kerberos authentication mechanism enables interoperability between other applications that support Kerberos authentication. It provides single sign on (SSO) end-to-end interoperable solutions and preserves the original requester identity.

About this task

In order to take advantage of Kerberos authentication with resources in the application server, you need to configure both DB2 and the application server to use Kerberos as the authentication mechanism. Read about Kerberos (KRB5) authentication mechanism support for security for a better understanding of how to set up Kerberos as the authentication mechanism in this version of the application server.

1. Configure Kerberos as the authentication mechanism for the application server.
   a. Configure the application server to use an LDAP user registry. See the topic on configuring Lightweight Directory Access Protocol user registries for more information.
   b. Configure Kerberos as the authentication mechanism. See the topic on configuring Kerberos as the authentication mechanism using the administrative console.
   c. Enable global security. See the topic on enabling security for more information.
2. Configure resource adapters for the application server to use Kerberos authentication.
   a. Create a JDBC Provider for DB2, and use it to create a data source. For more information on this task, refer to the topic on configuring a JDBC provider and data source.
   b. Configure the data source to connect to the desired DB2 server and data table.
   c. Configure the appropriate security settings for the data source.
      1) Set the authentication alias for XA recovery to **(none)**
      2) Set the component-managed authentication alias to **(none)**
      3) Set the mapping-configuration alias to **KerberosMapping**.

         Note: If the Kerberos credential connection fails, the resource adapter can be configured to try a second connection authentication using Default Principle Mapping. To configure this fallback, select an alias from the Container-managed authentication alias list. To disable this fallback, select **(none)** from the Container-managed authentication alias list.
      4) Configure the custom properties for the data source. In the **Additional Properties** → **Custom properties** panel for the data source, set the following properties to use Kerberos as the primary authentication method on a data source:

Name	Value
KerberosServerPrincipal	*domain_name* For example, `domain.rchland.ibm.com`
SecurityMechanism	11

3. Configure the settings for DB2 to use Kerberos authentication. For more information on how to configure DB2, see the link to the DB2 product family site in the IBM Suggests section.
   a. Configure the databases and tables that you will use for tests and general operation.
   b. Start a db2 command prompt.
   c. From the DB2 command prompt, set the authentication mechanism for DB2.
      1) To set the mechanism, enter the following:

         ```
         db2 update dbm cfg using authentication <mechanism>
         ```

         <mechanism> can be Kerberos or Server. Therefore, to configure Kerberos authentication, enter:

         ```
         db2 update dbm cfg using authentication Kerberos
         ```
      2) To validate that the authentication is set:
         a) Retrieve the configuration by entering the following command:

            ```
            db2 get db2 cfg
            ```

b) Scroll through the properties until you find the <Authentication> entry. Validate that this entry is set to the mechanism that you specified.

d. Configure the DB2 service. Start the db2 service that corresponds to your DB2 instance with a Kerberos user account that is not mapped to a SPN. This must be the same user account that you used when you configured LDAP security in the application server. To start a service with a defined user on Windows operating systems, for example:
   1) Open the Windows Services control panel. Click **Control Panel** → **Administrative Tools** → **Services**.
   2) Find the service that controls the DB2 installation.
   3) Right-click the service, and choose **Properties** from the list.
   4) In the new window, select the **Log On** tab.
   5) Select the **This account** radio button.
   6) Enter the user name and password for Kerberos User 2.
   7) Apply the changes, and click **OK**.
   8) Stop and start the DB2 service. If the user name and password is correct, then the service should start.
e. Configure java security.
   1) Create a jaas.conf file, and put it in the appropriate directory. On Windows operating systems, for example, you can put this file in c:\winnt
   2) Complete the jaas.conf file with the following:

```
JaasClient {
com.ibm.security.auth.module.Krb5LoginModule required
debug=false    useDefaultCcache=false;
};
```

   3) Locate the java directory structure that DB2 uses. Modify the java.security file, which is located in the java/jre/lib/security directory, to point to the jaas.conf file. For example:

```
login.config.url.1=file:c:/temp/jaas.conf
```

f. Verify that you can retrieve a ticket granting ticket (TGT). The DB2 machine requires a TGT from the Kerberos server. To manually attempt to retrieve a TGT, enter the following command at a prompt:

```
java DEBUG=TRUE com.ibm.security.krb5.internal.tools.Kinit –f –p
```

DB2 should now be fully configured with Kerberos.

Related concepts

Kerberos (KRB5) authentication mechanism support for security
The Kerberos authentication mechanism enables interoperability with other applications (such as .NET, DB2 and others) that support Kerberos authentication. It provides single sign on (SSO) end-to-end interoperable solutions and preserves the original requester identity.

Related tasks

Configuring Lightweight Directory Access Protocol user registries
To access a user registry using the Lightweight Directory Access Protocol (LDAP), you must know a valid user name (ID) and password, the server host and port of the registry server, the base distinguished name (DN) and, if necessary, the bind DN and the bind password. You can choose any valid user in the user registry that is searchable. You can use any user ID that has the administrative role to log in.

Configuring Kerberos as the authentication mechanism using the administrative console
You can use the administrative console to configure Kerberos as the authentication mechanism for the application server. When you have entered and applied the required information to the configuration, the Kerberos service principal name is formed as <*service name*>/<*fully qualified hostname*>@KerberosRealm, and is used to verify incoming Kerberos token requests.

Enabling security
By enabling security, you protect your server from unauthorized users and are then able to provide application isolation and requirements for authenticating application users.

"Configuring a JDBC provider and data source" on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

Related reference

"WebSphere Application Server data source properties" on page 1469
Use this page to set advanced data source properties in the application server. These properties activate and configure services that the application server applies to data sources to customize connections within an application server. These properties do not affect connections within the database.

Configuring Oracle Real Application Cluster (RAC) with the application server

Oracle Real Application Cluster (RAC) is a ″share-everything″ database architecture in which two or more Oracle RAC nodes are clustered together and share the same storage. The RAC nodes are connected together with a high-speed interconnect that enables fast communication between the Oracle nodes. The nodes can exchange various categories of data block ownership information during startup, lock information, exchange transaction information and data, and so on.

About this task

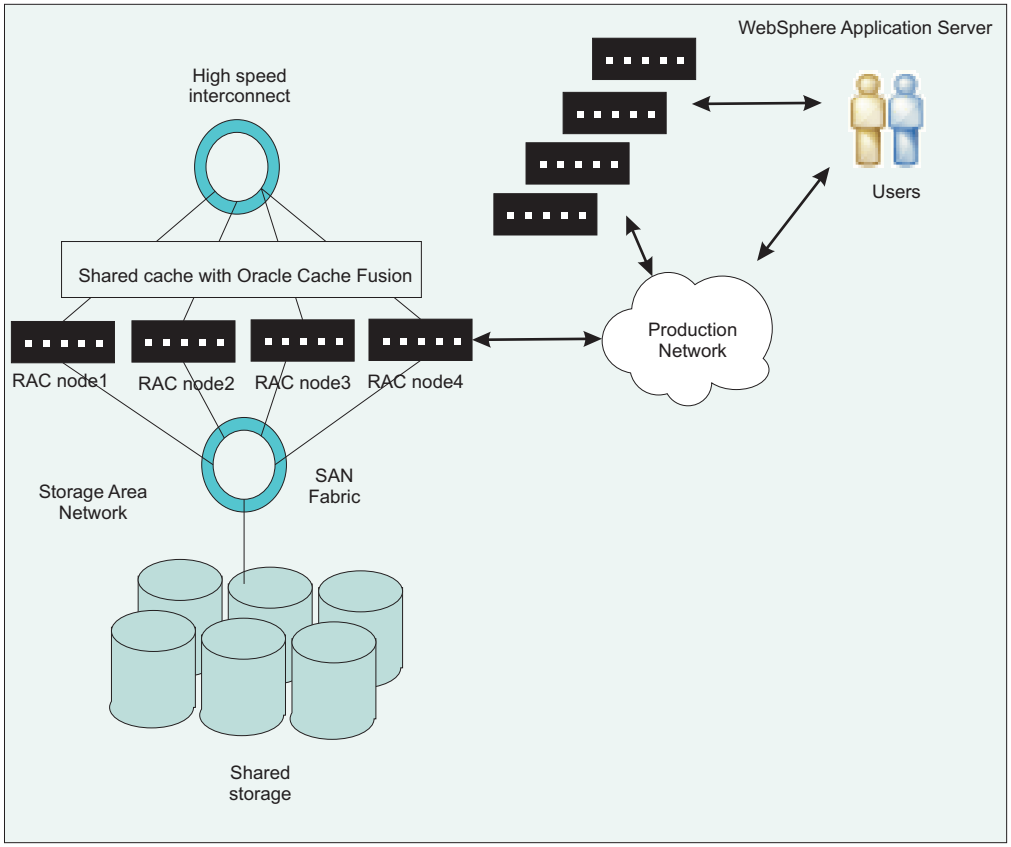
Note: Using the Oracle JDBC driver, you can configure failover support, load balancing, or both, in an Oracle RAC environment.

Oracle Real Application Clusters (RAC) is an option of an Oracle database that brings together two or more computers to form a clustered database that behaves as a single system. In a RAC database, Oracle processes that are running in separate nodes access the same data from a shared disk storage. First introduced in Oracle Version 9i, RAC provides both high availability and flexible scalability.

A typical Oracle RAC cluster consists of the following:

- **Cluster nodes** – 2 to *n* nodes or hosts, running the Oracle database server.
- **Network Interconnect** – a private network used for cluster communications and cache fusion. This is typically used for transferring database blocks between node instances.
- **Shared Storage** – used to hold the database system and data files. The shared storage is accessed by the cluster nodes.
- **Production network** – used by clients and application servers to access the database.

The following figure depicts a typical configuration for Oracle RAC:



Here are two of the many features that Oracle RAC provides:

- *Oracle Notification Service (ONS)* allows for Oracle RAC to communicate the status for the nodes, which are typically UP and DOWN events, to the Oracle JDBC driver and the driver's connection cache. To take advantage of ONS, you must configure the application server to use Oracle's connection caching instead of the application server's connection pooling feature. Read "Configuring Oracle connection caching in the application server" on page 1513 for more information on this process.
- *Distributed Transaction Processing (DTP)* is a feature that was introduced in Oracle 10gR2. When this feature is enabled, Oracle will ensure that all in-flight prepared transactions that belong to a DTP service for failed RAC instances are pushed to disk. Then, Oracle will restart the DTP service on any of the RAC instances that are still operational.

For more information on Oracle RAC and how it works with the application server, refer to *Building a high availability database environment using WebSphere middleware: Part 3: Handling two-phase commit in WebSphere Application Server using Oracle RAC* on the developerWorks web site.

- "Configuring a simple RAC configuration in an application server cluster."
- "Configuring Oracle connection caching in the application server" on page 1513.
- "Configuring two-phase commit distributed transactions with Oracle RAC" on page 1515.

Related reference

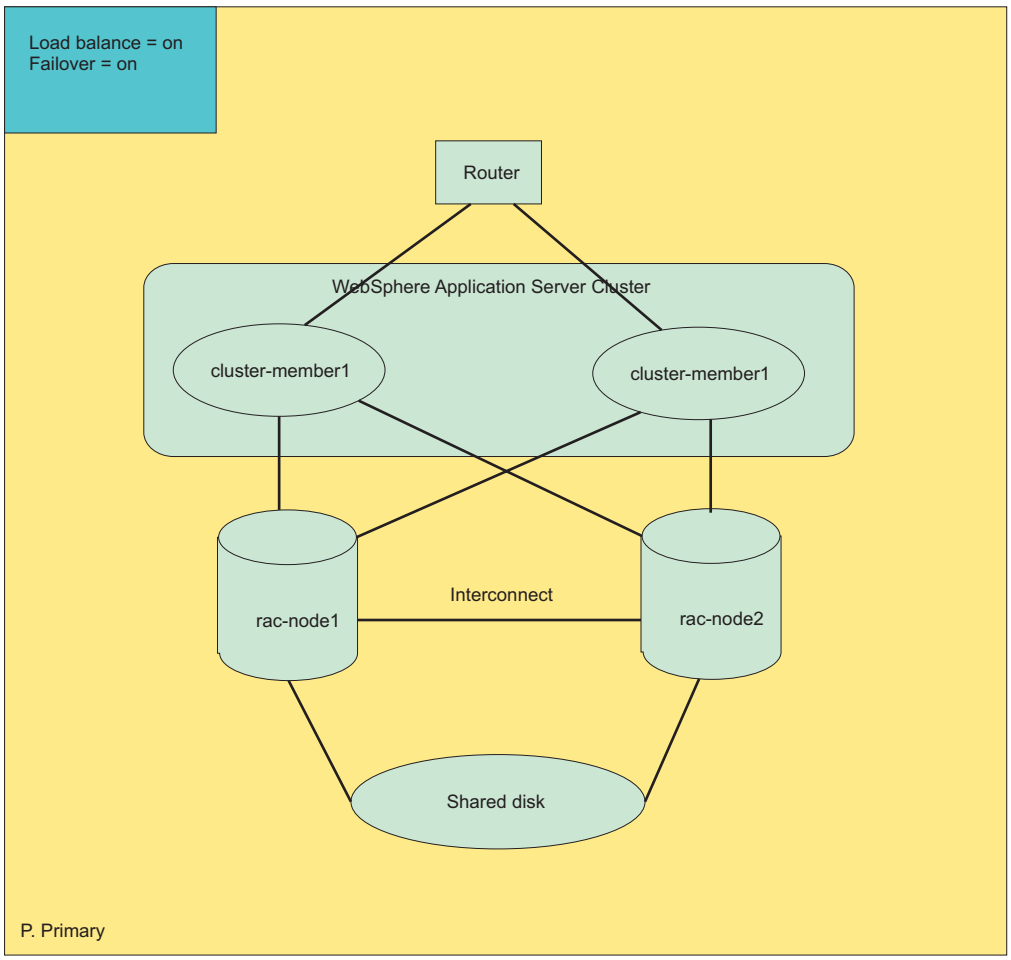
Building a high availability database environment using WebSphere middleware: Part 3: Handling two-phase commit in WebSphere Application Server using Oracle RAC

Configuring a simple RAC configuration in an application server cluster:

Oracle Real Application Cluster (RAC) is a ″share-everything″ database architecture that can provide high availability and load balancing. A typical configuration for an Oracle RAC contains two or more Oracle RAC nodes that are clustered together and share the same storage.

About this task

This figure depicts a typical RAC physical topology in a cluster environment for the application server, and both the failover and load balancing are enabled:



In the figure above, the application server cluster consists of two members: cluster-member1 and cluster-member2. The Oracle RAC physical configuration contains two nodes: rac-node1 and rac-node2. The RAC nodes can be located in the same physical machine with the cluster members, or they could be placed in entirely different machines. The actual placement does not impact the fundamental qualities of the services provided by RAC. To achieve both high availability and load-balancing, you can specify the Oracle data source URL for both cluster members in the application server with the required properties.

1. Navigate to the Oracle data source. Click **Resources** → **JDBC** → **Data sources** → ***oracle_data_source***. If you don't already have an Oracle data source, create a new data source by clicking **New** and completing the wizard. For the URL, substitute the properties in the next step.
2. Set the URL for the Oracle database with the required configuration parameters.

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=
                  (ADDRESS=(PROTOCOL=TCP)(HOST= rac-node1)(PORT=1521))
                  (ADDRESS=(PROTOCOL=TCP)(HOST= rac-node2)(PORT=1521))
                  (FAILOVER=on)(LOAD_BALANCE=on)
                  (CONNECT_DATA=(SERVER=DEDICATED)
                  (SERVICE_NAME=<service_name>)))
```

Note: Be aware of these configuration options:

- If you are not using Oracle services, then *service_name* will be the database name in the example. If you are using Oracle services, then *service_name* will be the name of the services.
- The example has FAILOVER and LOAD_BALANCE turned on. To turn one or both of these features off, change `on` to `off` in the above example.

3. Click **Apply** or **OK**.

Related tasks

"Configuring Oracle connection caching in the application server"
You can elect to configure an Oracle data source to use Oracle's connection caching feature instead of using the application server's connection pooling. Connection caching for Oracle databases is similar to connection pooling in the application server.

"Configuring two-phase commit distributed transactions with Oracle RAC" on page 1515
Real Application Cluster (RAC) configurations for Oracle 10g have an inherent issue with the transaction manager when Oracle attempts to recover two-phase commit distributed transactions that span over multiple Oracle RAC nodes. A problem can occur when one node fails, and Oracle opens up the other surviving nodes for business before the Oracle RAC completes the necessary recovery action for the node that has failed. The application server's ability to maintain transaction affinity provides you the ability to circumvent this issue.

"Configuring Oracle Real Application Cluster (RAC) with the application server" on page 1510
Oracle Real Application Cluster (RAC) is a ″share-everything″ database architecture in which two or more Oracle RAC nodes are clustered together and share the same storage. The RAC nodes are connected together with a high-speed interconnect that enables fast communication between the Oracle nodes. The nodes can exchange various categories of data block ownership information during startup, lock information, exchange transaction information and data, and so on.

Configuring Oracle connection caching in the application server:

You can elect to configure an Oracle data source to use Oracle's connection caching feature instead of using the application server's connection pooling. Connection caching for Oracle databases is similar to connection pooling in the application server.

About this task

Currently, Oracle only supports connection caching with data sources that use the oracle.jdbc.pool.OracleDataSource implementation class, instead of the oracle.jdbc.pool.OracleConnectionPoolDataSource or oracle.jdbc.xa.client.OracleXADataSource classes. By default, the Oracle JDBC providers in the application server are configured to use the oracle.jdbc.pool.OracleConnectionPoolDataSource for non-XA data sources, or oracle.jdbc.xa.client.OracleXADataSource for XA data sources. To enable Oracle connection caching, you must configure and use a new JDBC provider in the application server that implements the oracle.jdbc.pool.OracleDataSource class.

Note: Oracle connection caching does not support XA.

1. Create a data source and user-defined JDBC provider.
   a. Click **Resources** → **JDBC** → **Data sources**
   b. Select a server from the **Scope** drop-down list.

c. Click **New**.
d. Enter the name and JNDI name for the data source. Click **Next**.
e. Create a new JDBC provider. Select **Create new JDBC provider**, and click **Next**.
f. Define the required properties for the JDBC provider. Use the following configuration settings:
   - **Database type:** User-defined
   - **Implementation class name:** oracle.jdbc.pool.OracleDataSource

   Click **Next**.
g. Enter the class path for ojdbc6.jar, and click **Next**.
h. For **Data store helper class name**, enter com.ibm.websphere.rsadapter.Oracle11gDataStoreHelper. Click **Next**.
i. Define the security aliases for this data source, and click **Next**.
j. Finish the wizard.
k. Save the configuration changes.

2. Configure the data source that you created.
   a. Click the name of the data source. You will be taken to the configuration panel.
   b. Select **Custom properties**, and create or modify the properties for this data source. Enter or update the following custom properties:

Name	Value
disableWASConnectionPooling	true
connectionCachingEnabled	true
connectionCacheName	*your_cache_name*
removeExistingOracleConnectionPoolIfExists	true **Note:** The removeExistingOracleConnectionPoolIfExists property must be set to true so the application server will remove any existing Oracle connection pools with an identical name. Otherwise, the Oracle data source will fail the getConnection method if the pool name that is created has a name that is identical to an existing pool. For example, if you run a test connection, the test connection process will create an Oracle connection pool that will prevent the application server from working properly at run time.
URL	*Oracle_URL*

Note: The order in which the custom properties are set is important. This could be an issue, as the application server passes the properties as a collection and the order is not guaranteed. If you encounter this issue, contact Oracle and reference Oracle bug #6638862.

3. Click **Apply** or **OK**.
4. Save the changes to the application server's configuration.
5. Restart the application server.

Results

Be aware that Oracle will not display a message if the pool creation fails, and a normal connection will be returned instead. You can confirm that the connection pool is created by running your application, and try to issue a test connection. If the Oracle connection pool was created successfully, the test connection will fail with an exception that the Oracle pool name is already used. Therefore, you must create the data source and JDBC provider at the server scope.

Related tasks

“Configuring Oracle Real Application Cluster (RAC) with the application server” on page 1510
Oracle Real Application Cluster (RAC) is a ″share-everything″ database architecture in which two or more Oracle RAC nodes are clustered together and share the same storage. The RAC nodes are connected together with a high-speed interconnect that enables fast communication between the Oracle nodes. The nodes can exchange various categories of data block ownership information during startup, lock information, exchange transaction information and data, and so on.

“Configuring two-phase commit distributed transactions with Oracle RAC”
Real Application Cluster (RAC) configurations for Oracle 10g have an inherent issue with the transaction manager when Oracle attempts to recover two-phase commit distributed transactions that span over multiple Oracle RAC nodes. A problem can occur when one node fails, and Oracle opens up the other surviving nodes for business before the Oracle RAC completes the necessary recovery action for the node that has failed. The application server’s ability to maintain transaction affinity provides you the ability to circumvent this issue.

Configuring two-phase commit distributed transactions with Oracle RAC:

Real Application Cluster (RAC) configurations for Oracle 10g have an inherent issue with the transaction manager when Oracle attempts to recover two-phase commit distributed transactions that span over multiple Oracle RAC nodes. A problem can occur when one node fails, and Oracle opens up the other surviving nodes for business before the Oracle RAC completes the necessary recovery action for the node that has failed. The application server’s ability to maintain transaction affinity provides you the ability to circumvent this issue.

About this task

Errors can occur when the recovery process attempts to commit or rollback a transaction branch through a RAC node that was previously active but later failed. The transaction manager would receive the following exception:

```
ORA- 24756: transaction does not exist
```

If this error is encountered, the Oracle database administrator might need to manually resolve the in-doubt transaction by forcing a rollback or commit process. If you do not desire a manual intervention, however, you might want to configure an automatic and transparent strategy for transaction recovery.

If the in-doubt transaction is not resolved, any subsequent transactions will receive the following exception:

```
ORA-01591 lock held by in-doubt distributed transaction
```

The result is that portions of the database will not be usable.

The key to a transparent recovery strategy is to eliminate the possibility of a global transaction spanning more than one transaction branch over multiple RAC nodes. A transaction branch corresponds to a database connection that is enlisted in a global transaction. If all connections in a global two-phase commit transaction originate from the same node, transaction recovery problems should not arise. Configure an Oracle RAC with the application server to prevent errors with two-phase transactions.

The application server maintains transaction affinity for incoming connections, and you can take advantage of this feature to configure automatic recovery for Oracle RAC with two-phase commit transactions. If you implement this configuration, all connections from a given application server will be received from the same Oracle node, and the connections will finish on that same node. This configuration will avoid situations in which transactions span multiple nodes, and you should not experience a recovery problem if one or more Oracle nodes go down.

- You can elect to manually resolve the in-doubt transaction.
  1. Get the orphaned transaction ID. Issue the following command:

     ```
     sql > select state, local_tran_ID, Global_tran_Id from dba_2pc_pending where state = "prepared"
     ```

2. Roll back all of the transaction IDs that are in the prepared phase.

   ```
   sql > rollback force '';
   ```

- Configure an automatic strategy for transaction recovery.
  1. Create an Oracle service that has only one primary node. Creating the service with one primary node will ensure that load balancing is disabled. You can also specify one or more alternate nodes with the -a parameter. Run this command to create the service:

     ```
     srvctl add service -d <database_name> -s <service_name> -r <primary nodes> -a <alternate_nodes>
     ```
  2. Enable Distributed Transaction Processing (DTP) on the Oracle service. DTP was first introduced in Oracle 10gR2. Each DTP service is a singleton service that is available on only one Oracle RAC instance. Run this command:

     ```
     execute dbms_service.modify_service (service_name => '<service_name>' , dtp => true);
     ```
  3. Configure each cluster member in the application server to use the Oracle DTP service.

Results

If you configured an automatic recovery strategy, the DTP service will start automatically on the preferred instance. However, if the database is restarted, the DTP service will not start automatically. You can start the DTP service using this command:

```
srvctl start service -d  -s
```

If a RAC node stops working, Oracle will not failover the DTP service until the Oracle RAC cleanup and recovery is complete. Even if the Oracle nodes come back up, the Oracle DTP service will not return to the freshly restarted RAC node. Instead, you will have to manually move the service to the restarted RAC node.

When you configure DTP on the Oracle service, you have transferred load balancing from the Oracle JDBC provider to the application server. The workload will be distributed by the application server instead of Oracle, which is why you created services that do not implement load balancing and only use one primary node. This configuration prevents situations in which transaction processes span multiple RAC nodes and alleviates recovery problems that can arise when one or more RAC nodes fail.

Related tasks

"Configuring Oracle connection caching in the application server" on page 1513
You can elect to configure an Oracle data source to use Oracle's connection caching feature instead of using the application server's connection pooling. Connection caching for Oracle databases is similar to connection pooling in the application server.

"Configuring Oracle Real Application Cluster (RAC) with the application server" on page 1510
Oracle Real Application Cluster (RAC) is a ″share-everything″ database architecture in which two or more Oracle RAC nodes are clustered together and share the same storage. The RAC nodes are connected together with a high-speed interconnect that enables fast communication between the Oracle nodes. The nodes can exchange various categories of data block ownership information during startup, lock information, exchange transaction information and data, and so on.

Configuring client reroute for applications that use DB2 databases

The client reroute feature enables you to configure your client applications for a DB2 universal database to recover from a communication loss, and the applications can continue to work with minimal interruption. Rerouting is central to the support of continuous operations, but rerouting is only possible when there is an alternate location that is identified to the client connection.

Before you begin

This task assumes the following:

- You have a DB2 data source defined in the application server.
- The DB2 data source to which your application connects is running one of the following:

  - DB2 for z/OS Version 9.1 or later
  - DB2 Version 9.5 or later of DB2 for Linux, HP-UX, Solaris, or Windows.
- You have implemented the DB2 database with a redundant setup or the ability to fail the DB2 server to a standby node.

About this task

Note: Client reroute for DB2 allows you to provide an alternate server location, in case the connection to the database server fails. If you decide to use client reroute with the persistence option, the alternate server information will persist across Java Virtual Machines (JVMs). In the event of an application server crash, the alternate server information will not be lost when the application server is restored and attempts to connect to the database.

Without any configuration on the client side, a JDBC driver for DB2 supports the client reroute capability, if it is enabled, when the driver makes an initial connection to the DB2 server. When the JDBC driver connects to a DB2 server that has an alternate server configured, the primary server sends information about the alternate server to the JDBC driver. If the connection to the primary server fails, the JDBC driver is able to reroute connections to the alternate server. If the client process crashes, however, the alternate server information is lost, and the client will need to connect to the primary server again. If the client cannot make an initial connection to the primary server, the client will have no knowledge of the alternate server and cannot reroute.

You can configure a DB2 data source in the application server with the **Alternate server name** and **Alternate port number** fields, or with the clientRerouteAlternateServerName and clientRerouteAlternatePortNumber data source custom properties, to support client reroute even on the initial connection attempt. If the JDBC driver is not able to connect to the primary DB2 server, the information that is necessary for a client reroute is already present, and the JDBC driver can reroute the connection to an alternate server.

Additionally, if you have configured a DB2 data source as a Type 4 JDBC driver, you can use the **Client reroute server list JNDI name** field, or the clientRerouteServerListJNDIName data source custom property, to enable persistence of the client reroute state. Typically, when a connection is rerouted and the JDBC driver has connected to the alternate DB2 server, the alternate server sends information about its own alternate server to the JDBC driver. The JDBC driver will then have the information that is required to reroute the connection again if the alternate DB2 server is not available. Effectively, the server that was originally the alternate server is now the primary server, and a new alternate server has been established. If you enable persistence for client reroute, this new state can be remembered. If the application server crashes and is restarted, the JDBC driver can connect to the DB2 server that was considered the primary server at the time of the crash. Without the persistence feature, the JDBC driver would have to start from the original server configuration and attempt to connect to the server that was originally considered the primary server.

You can use the automatic client rerouting feature within the following DB2 configurable environments:

- Enterprise Server Edition (ESE) with the data partitioning feature (DPF)
- Data Propagator (DPROPR)-style replication
- High availability cluster multiprocessor (HACMP™)
- High availability disaster recovery (HADR).

1. In the administrative console, click **Resources** → **JDBC** → **Data sources** → ***data_source***.
2. Click **WebSphere Application Server data source properties**.
3. In the **DB2 automatic client reroute options** section, fill in the fields to enable client rerouting. Complete the following fields:

Alternate server names
Specifies the list of alternate server name or names for the DB2 server. If more than one alternate server name is specified, the names must be separated by commas. For example:

```
host1,host2
```

Alternate port numbers
Specifies the list of alternate server port or ports for the DB2 server. If more than one alternate server port is specified, the ports must be separated by commas. For example:

```
5000,50001
```

Note: Ensure that an equal number of entries must be specified for both alternate ports and hosts. Otherwise, a warning is displayed and client reroute is not enabled.

4. Optional: Enable client reroute with the persistence option.
   a. Complete the field for **Client reroute server list JNDI name**. The field specifies the JNDI name that is used to bind the DB2 client reroute server list into the JNDI name space. The DB2 database server will use this name to look up the alternate server name list when the alternate server information is not already in memory.

      Note: Be aware of the following:
      - This option is not supported for Type 2 data sources. If you use a DB2 data source that is configured as a Type 2 JDBC driver, the JDBC driver uses a catalog to persist the client reroute information. If this property is configured with a Type 2 driver, the application server will issue a warning.
      - Use different JNDI names among different data sources. Otherwise, when you delete a data source, and the JNDI entry is removed from the name space, the other data sources that share the JNDI entry will be affected.

5. Configure the retry count and interval for the client reroute function. Complete these two fields:

   Retry interval for client reroute
   Specifies the amount of time, in seconds, between retries for automatic client reroute.

   Maximum retries for client reroute
   Specifies the maximum number of connection retries that are attempted by the automatic client reroute function if the primary connection to the server fails. The property is only used when **Retry interval for client reroute** is set.

6. Click **OK**.
7. Restart the application server.

What to do next

If you later want to remove the client reroute information that is bound in JNDI, you can do so by deleting the data source. You can also use the unbind feature with the test connection service to delete the JNDI binding for the client reroute function from the application server's JNDI name space without deleting the data source.

To delete the JNDI binding for client reroute:

1. Select **Unbind client reroute list from JNDI**.
2. Click **OK**.
3. Save the configuration.
4. Click **Test connection** for the data source.
5. Deselect **Unbind client reroute list from JNDI**.
6. Click **OK**.
7. Save the configuration.

Related tasks

“Configuring a JDBC provider and data source” on page 1431
For access to relational databases, applications use the JDBC drivers and data sources that you configure for the application server.

Developing a custom DataStoreHelper class

Apply the WebSphere extension, GenericDataStoreHelper class, to create your own data store helper for data sources that the application server does not support. With this helper class, your JDBC configuration can use database-specific functions during transactions.

Before you begin

If you are using a configuration with a data source that is not supported by the application server, you might want to create a custom data store helper. This helper will allow you to leverage the database to perform functions during a transaction that would not otherwise be available. You will need to create a user-defined DataStoreHelper class, and there is information for creating a new exception handler to catch any exceptions that might be created with the use of your custom data handler.

1. Create a class that extends the com.ibm.websphere.rsadapter.GenericDataStoreHelper.java class or any of the existing data store helpers. Use the following code as an example; this type of data source is based on a user-defined JDBC provider:

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import javax.resource.ResourceException;

import com.ibm.websphere.appprofile.accessintent.AccessIntent;
import com.ibm.websphere.ce.cm.*;
import com.ibm.websphere.rsadapter.WSInteractionSpec;

/**
* Example DataStoreHelper class, demonstrating how to create a user-defined DataStoreHelper.
* The implementation for each method is provided only as an example.  More detail is probably
* required for any custom DataStoreHelper that is created for use by a real application.
* In this example, we will override the doStatementCleanup(),getIsolationLevel(), and set userDefined
* exception map.
*/
public class ExampleDataStoreHelper extends com.ibm.websphere.rsadapter.GenericDataStoreHelper
{

    public ExampleDataStoreHelper(java.util.Properties props)
    {
        super(props);

        // Update the DataStoreHelperMetaData values for this helper.
        getMetaData().setGetTypeMapSupport(false);

        // Update the exception mappings for this helper.
        java.util.Map xMap = new java.util.HashMap();

        // Add an Error Code mapping to StaleConnectionException.
        xMap.put(new Integer(2310),  StaleConnectionException.class);
        // Add an Error Code mapping to DuplicateKeyException.
        xMap.put(new Integer(1062),  DuplicateKeyException.class);
        // Add a SQL State mapping to the user-defined ColumnNotFoundException
        xMap.put("S0022",            ColumnNotFoundException.class);
        // Undo an inherited StaleConnection SQL State mapping.
        xMap.put("S1000",            Void.class);

        setUserDefinedMap(xMap);

        // If you are extending a helper class, it is
        // normally not necessary to issue 'getMetaData().setHelperType(...)'
        // because your custom helper will inherit the helper type from its
        //  parent class.


         }

    public void doStatementCleanup(java.sql.PreparedStatement stmt) throws SQLException
    {
        // Clean up the statement so it may be cached and reused.

        stmt.setCursorName("");
        stmt.setEscapeProcessing(true);
        stmt.setFetchDirection(java.sql.ResultSet.FETCH_FORWARD);
        stmt.setMaxFieldSize(0);
        stmt.setMaxRows(0);
```

```
        stmt.setQueryTimeout(0);
    }



    public int getIsolationLevel(AccessIntent intent) throws ResourceException
    {
        // Determine an isolation level based on the AccessIntent.

        // set WebSphere default isolation level to TRANSACTION_SERIALIZABLE.
        if (intent == null) return java.sql.Connection.TRANSACTION_SERIALIZABLE;

        return intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_OPTIMISTIC ?
               java.sql.Connection.TRANSACTION_READ_COMMITTED :
               java.sql.Connection.TRANSACTION_REPEATABLE_READ;
    }
        public int getLockType(AccessIntent intent) {
           if ( intent.getConcurrencyControl() == AccessIntent.CONCURRENCY_CONTROL_PESSIMISTIC) {
              if ( intent.getAccessType() == AccessIntent.ACCESS_TYPE_READ ) {
                  return WSInteractionSpec.LOCKTYPE_SELECT;
              }
              else {
                  return WSInteractionSpec.LOCKTYPE_SELECT_FOR_UPDATE;
               }
           }
           return WSInteractionSpec.LOCKTYPE_SELECT;
        }
}
```

2. Optional: Create your own exception handler class. Use the following code as a guide:

```
package com.ibm.websphere.examples.adapter;

import java.sql.SQLException;
import com.ibm.websphere.ce.cm.PortableSQLException;

/**
* Example PortableSQLException subclass, which demonstrates how to create a user-defined
* exception for exception mapping.
*/
public class ColumnNotFoundException extends PortableSQLException
{
    public ColumnNotFoundException(SQLException sqlX)
    {
        super(sqlX);
    }
}
```

3. Compile the newly created DataStoreHelper class or classes. You will need the following JAR files in your classpath to compile them:
   - *app_server_root*/dev/JavaEE/j2ee.jar
   - *app_server_root*/dev/was_public.jar

     Note: was_public.jar contains classes from *app_server_root*/lib/rsahelpers.jar that are needed to compile a custom DataStoreHelper class.
   - *app_server_root*/plugins/com.ibm.ws.runtime.jar
   - If you are using a development environment, such as Eclipse, you need to set the above JAR files in your classpath to be able to compile. Then, create a JAR file of the project after you have finished editing your files (see the help documentation for your development environment for specific instructions).
   - If you do not have development environment, and you are using the javac compiler:
     a. Create your .java file that extends the GenericDataStoreHelper or any other data store helper, as shown in Step 1.
     b. Change to your home directory after you are done editing your file or files in the command line utility.
     c. Set the classpath using this command:
     d. Compile your class or classes. For example, on Windows operating systems enter the following command (this will compile all the .java files in the directory that you specify):

        ```
        C:\javac your_directory\*.java
        ```

     e. From the Java directory, create a JAR file of all the compiled class files in your directory. For example, enter the following command on Windows operating systems (change *myFile* to the name you want for your JAR file):

```
C:\Java> jar cf myFile.jar *.class
```

For more information on using the javac compiler go to the website for the Java compiler at Sun Microsystems.

4. Place your compiled JAR files in a directory, and update the class path for the JDBC provider to include that location. For example, if your JAR file is c:\myFile.jar, then make sure to modify the JDBC class path to include c:\myFile.jar.
   a. Click **Resources** → **JDBC** → **JDBC Providers** > ***JDBC_provider***.
   b. In the **Class path** field, add the location of the JAR files that you compiled. For example, press ENTER in the field and add a new line:
      ```
      c:\myFile.jar
      ```
5. Configure the application server to use your new custom DataStoreHelper class.
   a. From the administrative console select **Resources** > **JDBC** > **Data Sources**.
   b. Select the data source that you want to configure with your custom DataStoreHelper class.
   c. In the section labeled **Data store helper class name**, select **Specify a user-defined data store helper**.
   d. Enter the class name for the data store helper that you created.
   e. Apply your changes and select **OK**.

Verifying a connection

Many connection problems can be easily fixed by verifying some configuration parameters. There are some steps that you must complete to enable a successful connection.

About this task

If your connection is still not successful after completing these steps and reviewing the applicable information, check the SystemOut.log for warning or exception messages. Then use the technical support search function to find known problems. See the IBM Suggests section of this article for a link to the IBM support and downloads page, which contains the search function.

1. Create the authentication data alias.
2. Create the JDBC provider.
3. Create a data source.
4. Save the data source.
5. If you created a new authentication alias, restart the server for which you need to verify connectivity.
6. Test the connection

   You can test your connection from the data source collection view or the data source details view. Access either view in the administrative console, and then select a connection from the list. Click the **Test Connection** button on the connection.

Test connection service

WebSphere Application Server provides a test connection service for validating data source configurations. The testConnection operation instantiates the data source configuration, gets a connection, and then immediately closes the connection.

If you associate your data sources with WebSphere variables, see the Creating, editing, and deleting WebSphere variables topic to verify that you configure them correctly. A `variable cannot be found` exception results from attempted use of a data source that is invoked through an incorrectly defined variable.

Activating the test connection service

There are three ways to activate the test connection service: through the administrative console, the wsadmin tool, or a Java stand-alone program. Each process invokes the same methods on the same MBean.

Administrative console

WebSphere Application Server allows you to test a connection from the administrative console by simply pushing a button: the *Data source collection*, *Data source settings*, *Version 4 data source collection*, and *Version 4 data source settings* pages all have **Test Connection** buttons. After you define and save a data source, you can click this button to ensure that the parameters in the data source definition are correct. On the collection page, you can select several data sources and test them all at once. Note that there are certain conditions that must be met first. For more information, see Testing a connection with the administrative console.

Note: The following exception occurs when you click Test Connection to connect a Sybase data source from the administrative console.

```
Test connection failed for data source isagent on server server1 at node
svtaix24Node01 with the following exception: java.lang.Exception:
java.sql.SQLException: JZ006: Caught IOException: java.net.ConnectException: A
remote host refused an attempted connect operation.DSRA0010E: SQL State = JZ006,
Error Code = 0
```

This exception occurs when the Sybase data source port number is not matched to the port configured in Sybase server. The default port number is 5000. Check the port number of your Sybase server in the interfaces file under `/<sybase install directory>`.

WsAdmin tool

The wsadmin tool provides a scripting interface to a full range of WebSphere Application Server administration activities. Because the Test Connection functionality is implemented as a method on an MBean, and wsadmin can invoke MBean methods, wsadmin can be utilized to test connections to data sources. You have two options for testing a data source connection through wsadmin:

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object. For information, see Testing a connection using wsadmin.

You can also test a connection by invoking the MBean operation. Use Example: Testing data source connection using wsadmin as a guide for this technique.

Java stand-alone program

Finally, you can test a connection by executing the `testConnection()` method on the `DataSourceCfgHelper` MBean. This method allows you to pass the configuration ID of the configured data source. The Java program connects to a running Java Management Extensions (JMX) server to access the MBean. In a base installation of Application Server, you connect to the JMX server running in the application server, usually on port 8880.

The return value from this invocation is either 0, a positive number, or an exception. 0 indicates that the operation completed successfully, with no warnings. A positive number indicates that the operation completed successfully, with the number of warnings. An exception indicates that the test of the connection failed.

You can find an example of this code in Example: Test a connection using testConnection(ConfigID).

Testing a connection with the administrative console

After you have defined and saved a data source, you can click the **Test Connection** button to ensure that the parameters in the data source definition are correct.

Before you begin

About this task

You can select multiple data sources on the data source collection page and test them as a group. Be sure that the following conditions are met before using the Test Connection button:

- If you are testing a connection using a WebSphere Application Server Version 4.0 type of data source, ensure that the *user* and *password* information is set.
- Designate variables appropriately.
  - WebSphere Application Server automatically sets environment variables for JDBC driver class paths on the i5/OS platform. Application Server names the variable according to your driver, either ${OS400_NATIVE_JDBC_DRIVER_PATH} or ${OS400_TOOLBOX_JDBC_DRIVER_PATH}. In the administrative console, you designate the complete path location of your driver as the value of the environment variable. (See the “JDBC provider settings” on page 1455 article for more information.)
  - If you download the latest JTOpen version of the jt400.jar file, use one of two placement strategies to keep the value of your environment variable accurate. You can place the file in the same directory that you specify for the ${OS400_TOOLBOX_JDBC_DRIVER_PATH} variable. Alternatively, you can place the jt400.jar file in a different directory and change the value of ${OS400_TOOLBOX_JDBC_DRIVER_PATH} to this different path.
- Restart the application server after you define or edit WebSphere variables.
- Restart the application server after you create or edit an authentication alias for the data source.
- You can now test a connection to the data source. On the data source collection page in the administrative console, select the data source and click **Test Connection** .

  A Test Connection operation can have three different outcomes, each resulting in a different message being displayed in the messages panel of the page on which you press the Test Connection button.
  1. The test can complete successfully, meaning that a connection is successfully obtained to the database using the configured data source parameters. The resulting message states: Test Connection for data source *DataSourceName* on process *ProcessName* at node *NodeName* was successful.
  2. The test can complete successfully with warnings. This means that while a connection is successfully obtained to the database, warnings were issued. The resulting message states: Test Connection for data source *DataSourceName* on process *ProcessName* at node *NodeName* was successful with warning(s). View the JVM Logs for more details.

     The **View the JVM Logs** text is a hyperlink that takes you to the JVM Logs console screen for the process.
  3. The test can fail. A connection to the database with the configured parameters is not obtained. The resulting message states: Test Connection failed for data source *DataSourceName* on process *ProcessName* at node *NodeName* with the following exception: *ExceptionText*. View the JVM Logs for more details.

     Again, the text for **View the JVM Logs** is a hyperlink to the appropriate logs screen.

Testing a connection using wsadmin

The *AdminControl* object of wsadmin has a testConnection operation that tests the configuration properties of a data source object.

Before you begin

About this task

The testConnection operation takes a data source *configuration ID* as an argument.

Note: This invocation cannot accept user IDs and passwords that must be defined in the database itself. This invocation can only be used for databases that do not require a user ID and password to make a connection (such as DB2 on a Windows machine), or for data sources that have a component-managed or container-managed authentication alias set on the data source object.

1. Invoke the getid() method for your data source.
2. Set the value of the *configuration id* to a variable.

   ```
   set myds [$AdminConfig getid /JDBCProvider:mydriver/DataSource:mydatasrc/]
   ```

   where */JDBCProvider:mydriver/DataSource:mydatasrc/* is the data source you want to test. After you have the configuration ID of the data source, you can test the connection to the database.
3. Test the connection to the database.

   ```
   $AdminControl testConnection $myds
   ```

Example: Testing a connection using testConnection(ConfigID):

The following sample code creates a data source instance and an associated connection instance, and tests them to ensure database connectivity.

This program uses JMX to connect to a running server and invoke the *testConnection* method on the *DataSourceCfgHelper* MBean. The acronym *ND* in a comment line indicates that the following code applies to WebSphere Application Server Network Deployment. The word *Base* in a comment line indicates that the following code applies to WebSphere Application Server.

```
/**
 * Description
 * Resource adapter test program to make sure that the MBean interfaces work.
 * Following interfaces are tested
 *
 *  ---  testConnection()
 *
 *
 * We need following to run
 * C:\src>java -Djava.ext.dirs=C:\WebSphere\AppServer\lib;C:\WebSphere\AppServer\java\jre\lib\ext testDSGUI
 * must include jre for log.jar and mail.jar, else get class not found exception
 *
 *
 */

import java.util.Iterator;
import java.util.Locale;
import java.util.Properties;
import java.util.Set;

import javax.management.InstanceNotFoundException;
import javax.management.MBeanException;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.RuntimeMBeanException;
import javax.management.RuntimeOperationsException;

import com.ibm.websphere.management.AdminClient;
import com.ibm.websphere.management.AdminClientFactory;
import com.ibm.ws.rsadapter.exceptions.DataStoreAdapterException;

public class testDSGUI {
```

```
//Use port 8880 for a Base installation or port 8879 for ND installation
String port = "8880";
// String port = "8879";
String host = "localhost";
final static boolean verbose = true;

// eg a configuration ID for DataSource declared at the node level for Base
private static final String resURI = "cells/cat/nodes/cat:resources.xml#DataSource_1";

// eg a 4.0 DataSource declared at the node level for Base
//    private static final String resURI = "cells/cat/nodes/cat:resources.xml#WAS40DataSource_1";

// eg Apache Derby DataSource declared at the server level for Base
//private static final String resURI = "cells/cat/nodes/cat/servers/server1/resources.xml#DataSource_6";

// eg node level DataSource for ND
//private static final String resURI = "cells/catNetwork/nodes/cat:resources.xml#DataSource_1";

// eg server level DataSource for ND
//private static final String resURI = "cells/catNetwork/nodes/cat/servers/server1:resources.xml#DataSource_4";

// eg cell level DataSource for ND
//private static final String resURI = "cells/catNetwork:resources.xml#DataSource_1";

 public static void main(String[] args) {
  testDSGUI cds = new testDSGUI();
  cds.run(args);
 }

/**
 * This method tests the ResourceMbean.
 *
 * @param args
 * @exception Exception
 */
 public void run(String[] args) {

  try {

 System.out.println("Connecting to the application server.......");

       /****************************************************************************/
       /**    Initialize the AdminClient                                          */
       /****************************************************************************/
 Properties adminProps = new Properties();
 adminProps.setProperty(AdminClient.CONNECTOR_TYPE, AdminClient.CONNECTOR_TYPE_SOAP);
 adminProps.setProperty(AdminClient.CONNECTOR_HOST, host);
 adminProps.setProperty(AdminClient.CONNECTOR_PORT, port);
 AdminClient adminClient = null;
 try {
     adminClient = AdminClientFactory.createAdminClient(adminProps);
 } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
       System.out.println("NLS: Cannot make a connection to the application server\n");
     ce.printStackTrace();
     System.exit(1);
 }

       /****************************************************************************/
       /**    Locate the Mbean                                                    */
       /****************************************************************************/
 ObjectName handle = null;
 try {
               // Send in a locator string
               // eg for a Base installation this is enough
               ObjectName queryName = new ObjectName("WebSphere:type=DataSourceCfgHelper,*");
```

```
               // for ND you need to specify which node/process you would like to test from
               // eg run in the server
//ND: ObjectName queryName = new OjectName
    ("WebSphere:cell=catNetwork,node=cat,process=server1,type=DataSourceCfgHelper,*");
               // eg run in the node agent
//ND: ObjectName queryName = new ObjectName
    ("WebSphere:cell=catNetwork,node=cat,process=nodeagent,type=DataSourceCfgHelper,*");
//ND: eg run in the Deployment Manager
//ND: ObjectName queryName = new ObjectName
   ("WebSphere:cell=catNetwork,node=catManager,process=dmgr,type=DataSourceCfgHelper,*");
 Set s = adminClient.queryNames(queryName, null);
 Iterator iter = s.iterator();
 while (iter.hasNext()) {
               // use the first MBean that is found
   handle = (ObjectName) iter.next();
  System.out.println("Found this ->" + handle);
  }
  if (handle == null) {
   System.out.println("NLS: Did not find this MBean>>" + queryName);
   System.exit(1);
  }
 } catch (MalformedObjectNameException mone) {
  System.out.println("Check the program variable queryName" + mone);
 } catch (com.ibm.websphere.management.exception.ConnectorException ce) {
  System.out.println("Cannot connect to the application server" + ce);
 }

        /*********************************************************************/
        /**           Build parameters to pass to Mbean                     */
        /*********************************************************************/
 String[] signature = { "java.lang.String" };
 Object[] params = { resURI };
 Object result = null;

       if (verbose) {
  System.out.println("\nTesting connection to the database using " + handle);
 }

 try {
              /*********************************************************************/
              /**  Start to test the connection to the database                   */
              /*********************************************************************/
  result = adminClient.invoke(handle, "testConnection", params, signature);
 } catch (MBeanException mbe) {
  // ****** all user exceptions come in here
  if (verbose) {
   Exception ex = mbe.getTargetException(); // this is the real exception from the Mbean
   System.out.println("\nNLS:Mbean Exception was received contains " + ex);
   ex.printStackTrace();
   System.exit(1);
  }
 } catch (InstanceNotFoundException infe) {
  System.out.println("Cannot find " + infe);
 } catch (RuntimeMBeanException rme) {
  Exception ex = rme.getTargetException();
  ex.printStackTrace(System.out);
  throw ex;
 } catch (Exception ex) {
  System.out.println("\nUnexpected Exception occurred: " + ex);
  ex.printStackTrace();
 }

        /*********************************************************************/
        /**  Process the result.  The result will be the number of warnings   */
        /**  issued.  A result of 0 indicates a successful connection with    */
        /**  no warnings.                                                     */
        /*********************************************************************/
```

```
 //A result of 0 indicates a successful connection with no warnings.
 System.out.println("Result= " + result);

  } catch (RuntimeOperationsException roe) {
 Exception ex = roe.getTargetException();
 ex.printStackTrace(System.out);
  } catch (Exception ex) {
 System.out.println("General exception occurred");
 ex.printStackTrace(System.out);
  }
 }
}
```

Configuring data access for the Application Client

Configuring data access for the Application Client involves specifying the resource reference and associated database information required for data access. This specification is done as part of the assembly and deployment steps for the Application Client.

About this task

There are two tools needed to configure data sources used by J2EE application clients:

- An assembly tool for defining the resource reference in the deployment descriptor; and
- The Application Client Resource Configuration Tool (ACRCT) for defining the connection to the database in the client deployment environment.

Data access from an application client uses the JDBC driver connection functions directly from the client side. It does not take advantage of the additional pooling support available in the WebSphere Application Server run time. Configuring data access for an application client does not require configuration of a JDBC provider and data source on the WebSphere Application Server server machine.

If you want to take advantage of the pooling and additional database functions provided by the product, it is recommended that your client application utilize an enterprise bean running on the server side to perform data access.

Defining an application client resource reference using an assembly tool

1. Assemble your application client module as described in Assembling application clients.
2. Create a new resource reference:
   a. In a Project Explorer view, right-click your application client module and click **Open With > Deployment Descriptor Editor**.
   b. On the **References** tab, click **Add > Resource reference > Next**.
   c. On the Resource Reference page, enter the **Name** of this resource reference. The Application Client for WebSphere Application Server run time uses this name for two purposes: to bind the object into the *java:comp/env* portion of the JNDI namespace, and to find client specific configuration information. If the code for the Application Client performs a lookup for *java:comp/env/jdbc/myDB*, the name of the resource reference should be *jdbc/myDB*.
   d. For **Type**, select *javax.sql.DataSource* for JDBC connections.
   e. For **Authentication**, select *Application* if your client application intends to provide authentication information. If the Application Client run time provides the authentication information (as configured by the Application Client Resource Configuration tool), select *Container*.
   f. Ignore the **Sharing scope** setting; it is unused in an application client resource reference. All Application Client resources are not shared.
   g. Click **Finish**.
   h. Close the deployment descriptor and save your changes.

Results

The JNDI name field appears under **WebSphere Bindings** after your add the reference.

Client configuration with the ACRCT
About this task

There are two client resources for you to configure in the Application Client Resource Configuration Tool (ACRCT) to enable data access from an application client: a data source provider and a data source.

Note: The following objects, which can be bound into the server name space, are not supported on the client:
- Java 2 Connector (J2C) objects
- Connection manager objects

The Application Client does not provide client database drivers. If your client application uses a database directly, rather than using an enterprise bean, you must provide the database drivers on the client machine. This action can involve contacting your database vendor to acquire client database driver code and licenses.

Instead of accessing the database directly, it is recommended that your client application use an enterprise bean. Accessing a database through an enterprise bean eliminates the need to have database drivers on the client machine because the database access is handled by the enterprise bean running on the application server. Enterprise beans can also take advantage of the additional database functions provided by the run time.

1. Configure a new data source provider as described in Configuring new data source providers. This provider describes the JDBC database implementation for your client application.
2. Enter the following information on the **General** tab:
   a. A **name** for this data source provider.
   b. Optional: A **description**.
   c. The **classpath** to the data source provider implementation classes or JAR files. This is optional if the implementation classes or JAR files are already in the class path configuration of the client.
   d. The name of the **implementation class**. For example, for DB2 this value is *COM.ibm.db2.jdbc.DB2DataSource*. Remember this class must implement the *javax.sql.DataSource* class. The ACRCT does not verify this class and you receive an error when you run your client application if the class does not implement *javax.sql.DataSource*.

   Use the **Custom** tab to configure non-standard properties of the data source provider. This panel enables you to enter property-value pairs. During run time the *implementation class name* is created and any custom properties added on this panel are set on the newly created data source object using reflection. Any properties configured on this panel must have an appropriate set method on the data source class. For example, assume there is a property called *use2Phase* and its value should be 1. On the custom panel you enter the value *use2Phase* into the **name** column and the value *1* into the **value** column. The Application Client run time then uses reflection to find a property on the data source class called, typically *setUse2Phase* and call that method passing the value of 1. See your database product documentation for valid properties on your data source implementation.
3. Click **OK**.
4. Configure a new data source as described in Configuring new data sources for application clients. This describes the client properties of the database your client application uses.
5. Enter the following information on the **General** tab:
   a. A **Name**. This field is required and identifies a name for the Application Client Resource Configuration Tool to use. This name is **not** used by your client application program.
   b. Optional: A **description**.

c. The **JNDI name**. This field is required and must match the value entered in the **Name** field on the Add Resource Reference page of the assembly tool. In the example above, set this value to *jdbc/myDB*.

d. Optional: The **Database Name**.

e. Optional: Your *userid* in the **User** field.

f. Optional: Your *password* in the **Password** field. This password does not display.

g. Your password again to confirm in the **Re-Enter password** field. Note: The **User** and **Password** fields are used only when the **Authentication** field on the Add Resource Reference page of the assembly tool is set to *Container*.

Resource references

Use this page to designate how the resource references of application modules map to the actual resources that are configured for the application.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Resource references**.

- If your application uses any of the following resource types, you can set or reset their mapping configurations:
  - Default messaging JMS queues destinations
  - Default messaging JMS topic destinations
  - Data source
  - Generic JMS connection factory
  - Mail session
  - J2C connection factory
  - JMS queue connection factory for the JMS provider of WebSphere MQ
  - JMS queue destination for WebSphere MQ
  - JMS topic connection factory for WebSphere MQ
  - JMS topic destination for WebSphere MQ
  - Unified JMS connection factory for WebSphere MQ
  - URL configuration
- The page is comprised of sections that correspond to each applicable resource type. Each section heading is the class name for the resource. If your application contains only one applicable resource type, you see only one section.
- Each section contains a table. Each table row depicts a resource reference within a specific module of your application.
- The rows contain the JNDI names of resource mapping targets for your references *only* if you bound them together during application assembly. You can modify those bindings on this administrative console page.
- To set your mappings:
  1. Select a row. Be aware that if you check multiple rows on this page, the resource mapping target that you select in step 2 applies to all of those references.
  2. Click **Browse** to select a resource from the new page that is displayed, the Available Resources page. The Available Resources page shows all resources that are available mapping targets for your application references.
  3. Click **Apply**. The console displays the Resource references page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
  4. Repeat the previous steps as necessary.
  5. Click **OK**. You now return to the general configuration page for your enterprise application.

- **For data sources and connection factories:** Sections for these resource types contain an additional set of steps for modifying your security settings. Use the last column in the displayed table to view the authorization type for each resource configuration per application module. You can modify the corresponding authentication method only if the authorization type is container. Container-managed authorization indicates that the product performs signon to the resource rather than the enterprise bean code. The reconfiguring process differs slightly for each authentication method option:
  - If you select **None**:
    1. Determine which resource configurations to designate with no authentication method.
    2. Select the appropriate table rows.
    3. Select **None** from the list of authentication method options that precede the table.
    4. Click **Apply**.
  - If you select **Default**:
    1. Determine which resources to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. You must apply this option to each resource individually if you want to designate different authentication data aliases. See the ″J2EE connector security″ Information center topic for more information on the default mapping configuration.
    2. Select the appropriate table rows.
    3. Select **Use default method** from the list of authentication method options that precede the table.
    4. Select an authentication data entry or alias from the list.
    5. Click **Apply**.
  - If you select **Use trusted connections (one-to-one mapping)**:
    1. You must have a data source server that is running DB2 Version 9.1 for z/OS, and the data source must have trusted context enabled.
    2. Select an authentication alias from the list that matches an alias that is already defined in the DB2 data source. If you do not have an alias defined that is suitable, you need to define a new alias.
    3. Select a data source from the table that has trusted context enabled.
    4. Click **Apply**.
    5. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.
  - If you select **Custom login configuration**:
    1. Determine which resources to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the ″J2EE connector security″ Information center topic for more information on custom JAAS login configurations.
    2. Select the appropriate table row.
    3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
    4. Select an application login configuration from the list.
    5. Click **Apply**.
    6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select
Select the check boxes of the rows that you want to edit.

Module
The name of a module in the application.

EJB
The name of an enterprise bean that is contained by the module.

URI
Specifies location of the module relative to the root of the application EAR file.

Resource Reference
The name of a resource reference that is used in the enterprise bean, if applicable, and is declared in the deployment descriptor of the application module.

Target Resource JNDI name
The Java Naming and Directory Interface (JNDI) name of the resource that is the mapping target of the resource reference.

Data type String

Login configuration
This column applies to data sources and connection factories only and refers to the authorization type and the authentication method for securing the resource.

Mapping-configuration alias
This panel allows you to select a mapping configuration alias for the resource that you are configuring. This panel is only available when security domains are defined. Security domains allow you to isolate mapping configuration aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them. For example, a cell-scoped security domain will be hidden from the tree if all servers and clusters in the tree have defined their own security domain. If you are looking for an alias that is not visible in the tree, it is because the alias cannot be used by any servers that have visibility to this resource. In this case, you must define the alias at the global scope or in a different security domain that is visible to this resource.

To view this administrative console panel:

1. You must have a security domain defined in the application server.
2. Click one of the following paths in the administrative console:
   - **Resources** → **JDBC** → **Data sources** → ***data_source***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.
   - **Resources** → **JDBC** → **JDBC Providers** → ***jdbc_provider*** → **Data sources** → ***data_source***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.
   - **Resources** → **Resource Adapters** → **J2C connection factories** → ***j2c_connection_factory***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.

Note: Be careful when selecting an alias, because it is possible to select an alias that is only accessible by a subset of the servers that will use the resource. If you select a global alias, you are guaranteed that an alias by that name will be accessible to all users of the resource. If the alias has been overridden in a security domain, however, that alias will be used instead of the global one. The tree view includes icons to help you select the proper alias:

-  The alias is accessible by all servers that can access this resource.
-  There is at least one server that cannot access the alias. Check the tree view to see if this is OK for the application that will be using this resource.

-  The alias is defined in multiple places.

Select a J2C authentication alias

This panel allows you to select a J2C authentication alias for the resource that you are configuring. This panel is only available when security domains are defined. Security domains allow you to isolate J2C authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them. For example, a cell-scoped security domain will be hidden from the tree if all servers and clusters in the tree have defined their own security domain. If you are looking for an alias that is not visible in the tree, it is because the alias cannot be used by any servers that have visibility to this resource. In this case, you must define the alias at the global scope or in a different security domain that is visible to this resource.

To view this administrative console panel:

1. You must have a security domain defined in the application server.
2. Click one of the following paths in the administrative console:
   - **Resources** → **JDBC** → **Data sources** → ***data_source***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.
   - **Resources** → **JDBC** → **JDBC providers** → ***jdbc_provider*** → **Data sources** → ***data_source***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.
   - **Resources** → **Resource Adapters** → **J2C connection factories** → ***j2c_connection_factory***. Click **Browse...** in the security section for **Component-managed authentication alias** or **Container-managed authentication alias**.
   - **Resources** → **Resource Adapters** → **J2C activation specifications** → ***j2c_activation_specification***. Click **Browse...** in the security section for **Authentication Alias**.
   - **Resources** → **JMS** → **Connection factories** → ***connection_factory***. Click **Browse...** in the security section for **Authentication Alias**.
   - **Resources** → **JMS** → **JMS providers** → ***jms_provider*** → **[Additional properties] Connection factories** → ***connection_factory***. Click **Browse...** in the security section for **Authentication Alias**.

Note: Be careful when selecting an alias, because it is possible to select an alias that is only accessible by a subset of the servers that will use the resource. If you select a global alias, you are guaranteed that an alias by that name will be accessible to all users of the resource. If the alias has been overridden in a security domain, however, that alias will be used instead of the global one. The tree view includes icons to help you select the proper alias:

-  The alias is accessible by all servers that can access this resource.
-  There is at least one server that cannot access the alias. Check the tree view to see if this is OK for the application that will be using this resource.
-  The alias is defined in multiple places.

Considerations for isolated resource providers

There are some design considerations that you should be aware of when working with resource providers that you have specified to be isolated in their own class loaders.

Be aware of the following issues that you need to address if you isolate a resource provider in it's own class loader:

- **Client container**

The client container does not manage the class path of resource providers, so resource providers that are isolated will not be supported in the client container.

- **Multiple resource provider versions per application**

  If an application refers to resources from multiple versions or implementations of the same resource provider, then all of the resource providers that are referenced must be isolated.

- **References to isolated resource provider classes**

  If a module directly refers to classes that are loaded by an isolated resource provider, which means the module has import statements of resource provider classes, the following restrictions are in place:

  – The module can only refer to resources from one version or implementation of an isolated resource provider. This is an inherent class loading restriction, because a module class loader can only refer to one version of a class.

  – The module cannot perform direct JNDI lookup without the use of Java EE resource reference meta-data. This restriction is required, because without resource reference metadata the application server has no mechanism to link the class loader of the module to the class loader of the isolated resource provider.

  The relational resource adapter does not generally allow direct access to resource provider classes, so these restrictions will typically only affect modules that implement the com.ibm.websphere.rsadapter.WSCallHelper class. For mail providers, these restrictions will most likely be in place, because the javax.mail API relies heavily on classes rather than interfaces. Therefore, the implementation details are necessarily part of the API.

Configuring data access security

Safeguard access to your enterprise data by designating credentials that WebSphere Application Server uses to authenticate database connections.

Before you begin

Review the history and roles of the database tables in your environment; this information helps you determine the best approach for securing each table. You can either rely on the default authentication mechanism of a user profile, or override it by configuring a security scheme for the individual application component that requires access to the database. Generally, consider overriding the default profile security if your database tables were created prior to the Application Server installation, or if programs outside of Application Server also access the tables.

About this task

To authenticate database connections with user profile credentials, use an existing profile from any of three scopes in your application serving environment. Otherwise, consult the following Application component authentication table for methods of overriding user profile settings.

- **System-level** scope option: Change the default user profile of WebSphere Application Server to an existing profile in your i5/OS system that has authority to access the database tables.
- **WebSphere Application Server instance** scope option: Add authority for accessing the tables to the default Application Server user profile.
- **Individual application server** scope option: If you designated a user profile for an individual application server that is different from the profile of the entire WebSphere Application Server instance, add database authority to that new user profile.

  For database tables that are created by WebSphere Application Server and are used only within the Application Server environment, you generally do not need to change or override the default security. You can simply add authority for accessing the database to the WebSphere Application Server user profile.

- **Overriding user profile credentials:** In the following table, locate the type of application component for which you need to create authentication credentials. Choose a method from the Authentication

strategies column.

Table 13. Application component authentication

Type of object that requires access to backend data	The data is created by:	Library	Authentication strategies
Servlet session	WebSphere Application Server	By default, QWAS6SN. Specify a different library by editing the libraries custom property in the Session Manager persistence panel.	By default, the user profile under which the application server runs. Specify a different library by editing the libraries custom property in the Session Manager persistence panel.
Entity enterprise beans that use container managed persistence (CMP)	WebSphere Application Server or already exists	User-defined	By default, the user profile under which the application server runs. Specify a different user profile by creating a JAAS alias for the enterprise bean data source. See the ″Creating JAAS aliases″ note that follows this table for more information.
User-written data access code in servlets, session beans, and entity beans with bean-managed persistence (BMP), which might use connection pooling	WebSphere Application Server, user code, or already exists	User-defined	By default, the user profile under which the application server runs. Specify a different user profile by modifying the user code to explicitly use a user ID and password with the database connection. Alternatively, you can create a JAAS alias for the data source that you configure for your servlet or enterprise bean. See the ″Creating JAAS aliases″ note that follows this table for more information.

Note:

1. Consult the Managing Java 2 Connector Architecture authentication data entries article for instruction on creating a JAAS alias.
2. After following those steps, navigate to the relevant data source properties page in the administrative console: **Resources** > **Data sources** > *my_data_source*.
3. Designate the new JAAS credentials as the component-managed alias for the data source.
4. Restart the application server for the new security configuration to take effect.

Note: If you use component-specific authentication, be aware that some authentication strategies take precedence over others:

Enterprise bean

You can specify a data source for the enterprise bean and create a JAAS alias for association with that data source. The user ID and password properties of the alias control which user ID can access the tables that are defined by the data source. The JAAS alias that you associate with the enterprise bean data source takes precedence

over both the application server user ID and the ID of the EJB container data source. If you specify a data source for the enterprise bean but do not specify a JAAS alias for it, your i5/OS system uses the application server ID. The application server ID is the default, regardless of whether you specified a data source for the EJB container.

EJB container
You can specify a data source for the EJB container and create a JAAS alias for association with that data source. The user ID and password properties of the alias control which user ID can access the tables that are defined by the data source. The JAAS alias that you associate with the EJB container data source takes precedence over the application server user ID. However, if you do not specify a JAAS alias for the EJB container data source, your i5/OS system uses the application server ID to authenticate access to the database tables.

Application server
The application server runs under the user ID that is specified in the USER ID property for the application server. Any database tables that are accessed should allow access to the specified user ID. By default, the application server runs under the QEJBSVR user profile. Therefore, the database tables must allow access to the QEJBSVR user profile.

Passing client information to a database

Using a WebSphere Application Server API or trace function, you can pass unique client information on every connection that originates from the same data source.

About this task

Some databases, such as DB2, support a data source custom property that triggers your database servers to extract client information from WebSphere Application Server connections. (Consult the database documentation to see whether your product supports this capability and which property the product requires.) Be aware, however, that these properties introduce a very limited functionality in Application Server. Consequently, an application server connection manager incurs the following risky behaviors, which can result in the transfer of wrong client information to the database.

- The connection manager cannot change client information on the data source, or the connections obtained from that data source, dynamically.
- The connection manager must set the same client information on all connections that are obtained from that data source. For example, if you set `ApplicationName` as part of the data source `clientInformation` property, all connections from that data source have the same application name.

Application Server offers two methods of passing client information that provide the necessary connection management flexibility. Using either method, you can set client information on some connections and not others, as well as set different client information on different database connections from the same data source.

- Use the IBM proprietary *setClientInformation(Properties)* API. The API is defined on the *WSConnection* class, which is part of the *plugins_root*`/com.ibm.ws.runtime_6.1.0.jar` file.
  1. Cast the connection objects in your applications to *com.ibm.websphere.rsadapter.WSConnection* before calling the API.
  2. Optionally, set a properties object for adding new client information on a connection if and when new information is introduced by the backend database:

     ```
     public void setClientInformation (Properties props)throws SQLException;
     ```
- You can also activate the function implicitly (that is, within Application Server) by using the *WAS.clientinfo* trace string. Enable this trace dynamically, from the administrative console, just as you activate any other trace. For more information about passing client information implicitly, see “Implicitly set client information” on page 1536.

Results

Example

See “Example: Setting client information with the setClientInformation(Properties) API.”

Example: Setting client information with the setClientInformation(Properties) API

With this API, set WebSphere Application Server client information on connections to pass that information to your database.

The following example code calls setClientInformation(Properties) on the com.ibm.websphere.rsadapter.WSConnection object.

```
import com.ibm.websphere.rsadapter.WSConnection;
.....
try {
   InitialContext ctx = new InitialContext();
   //Perform a naming service lookup to get the DataSource object.
   DataSource ds = (javax.sql.DataSource)ctx.lookup("java:comp/jdbc/myDS");
  }catch (Exception e) {System.out.println("got an exception during lookup: " + e);}

  WSConnection conn = (WSConnection) ds.getConnection();
  Properties props = new properties();
  props.setProperty(WSConnection.CLIENT_ID, "user123");
  props.setProperty(WSConnection.CLIENT_LOCATION, "127.0.0.1");
  props.setProperty(WSConnection.CLIENT_ACCOUNTING_INFO, "accounting");
  props.setProperty(WSConnection.CLIENT_APPLICATION_NAME, "appname");
  props.setProperty(WSConnection.CLIENT_OTHER_INFO, "cool stuff");
  conn.setClientInformation(props);
  conn.close()
```

Parameters

props contains the client information to be passed. Possible values are:

- WSConnection.CLIENT_ACCOUNTING_INFO
- WSConnection.CLIENT_LOCATION
- WSConnection.CLIENT_ID
- WSConnection.CLIENT_APPLICATION_NAME
- WSConnection.CLIENT_OTHER_INFO
- WSConnection.OTHER_CLIENT_TYPE

Refer to the WSConnection documentation for more details on which client information is passed to the backend database. To reset the client information, call the method with a null parameter.

Exceptions

This API creates an SQL exception if the database issues an exception when setting the data.

Passing client info to a db cdat_clientinfo

Implicitly set client information

If you track client information in your database, you can choose one of two ways to pass WebSphere Application Server client data on database connections.

You can choose to *explicitly* pass the information on connections by calling an IBM proprietary API, setClientInformation(Properties), on the com.ibm.websphere.rsadapter.WSConnection object within your application code. The com.ibm.websphere.rsadapter.WSConnection object is located in the *plugins_root*/com.ibm.ws.runtime_6.1.0.jar file. In some cases, however, you might want WebSphere

Application Server to handle the passing of client information to database connections. This method of setting the client information is referred to as *implicit*. You might choose the implicit method because:

- You want to keep your application free of proprietary APIs, or
- Your application uses container-managed persistence (CMP), in which case you cannot use the proprietary API to set client information on database connections.

The WebSphere Application Server trace facility provides the capability for setting client information implicitly. You can designate one of two special trace groups to enable or disable client information passing: “WAS.clientinfo trace” or “WAS.clientinfopluslogging trace” on page 1538.

Possible run-time scenarios

- Connection sharing

  In the case of connection sharing, WebSphere Application Server sets the client information on the first acquired connection handle only. If connection sharing is enabled and two or more getConnection methods are called (resulting in two handles on the same connection), only the first getConnection call causes the client information to pass to the backend database. This scenario does not apply to the explicit process of passing client information; in such cases every setClientinformation method is relayed to the database regardless of connection sharing.

- Implicit/explicit co-existence

  When you use both the explicit and implicit procedures for relaying client information, some combination of the explicitly set data and implicitly set data is combined, but the explicit setting usually takes precedence. For example, if the application sets the client accounting information to ″myAccountingInfo″, the final accountingInfo string that is passed to the backend database looks something like the following sample code:

  ```
  000325_WSRdbManagedConnectionImpl@1234_myAccountingInfo:
  ```

  where `000325` is the thread id and `WSRdbManagedConnectionImpl@1234` is the WebSphere connection instance.

- Client information reset

  When you configure Application Server to pass client information, it does reset client information when a connection is returned to the pool, but *only* if the WAS.clientinfo and WAS.clientinfopluslogging trace mechanisms are disabled (that is, WAS.clientinfo=all=disabled:WAS.clientinfopluslogging=all=disabled).

  In the explicit case, however, the reset operation is done only when the application issues setClientInformation(null) on the WSConnection connection.

WAS.clientinfo trace

By default, the implicit mechanism is disabled. You can turn on this mechanism dynamically, without stopping and starting your application server, or statically by setting the WebSphere Application Server trace group *WAS.clientinfo=all=enabled*.

The information implicitly collected and set on the database connection consists of the *user name*, *user location* and *application name*.

Note: User name and user location can only be implicitly collected and set on the database connection if you enable Java 2 security.

user name
The name of the user that initiates the application request. This option is collected and passed to the backend database (when supported) only if Java 2 security is enabled. Information here is collected by calling the WSSecurityHelper.getFirstCaller method.

user location
The name of the location of the user, in the form of cell:node:server. This option is collected and

passed to the backend database, when appropriate, only when Java 2 security is enabled. Information here is collected by calling the WSSecurityHelper.getFirstServer method.

application name
The name of the application running. This value is the output of the getApplication method from the Java EEName object. This value is collected regardless of the Global Security setting.

WAS.clientinfopluslogging trace

When debugging database problems, such as deadlocks, there is a set of information that is needed to help with the debugging effort. This information is typically obtained by enabling a WebSphere Relational Resource Adapter (RRA) trace, and an Enterprise JavaBean (EJB) container trace. However, there are some cases where timing is an issue when reproducing a given problem. Having too much tracing information can alter the behavior of the application, such as change the timing, and the problem might no longer occur.

Because of this situation, a new trace group is provided where only a minimum set of information is collected. This trace group is WAS.clientinfopluslogging. This function sets the client information implicitly on the connection, just like the WAS.clientinfo trace, as well as logs and traces important application activities. Those activities are:

- SQL Strings that are run (such as, select userId from tabl1 where id=? for update).
- Start, commit, and rollback of transactions.
- EJB calls (such as, Create, Remove, findByPrimaryKey, and so on).

Setting client information traces with the administrative console

Use one of two WebSphere Application Server trace groups to pass client information to a database.

About this task

The WAS.clientinfo and WAS.clientinfopluslogging traces pass client data on every connection that originates from the same data source.

1. Open the administrative console.
2. Select **Troubleshooting**.
3. Select **Log and Trace**.
4. Select the server you want to use.
5. Select **Change log detail levels**.
6. Select the **Configuration** or **Run time** tab.
7. In the **Trace Specification** entry field, type `WAS.clientinfopluslogging=all`. This starts the WAS.clientinfopluslogging trace, which passes the client information to the backend data store (if your database supports receiving client information in this manner) and logs important client activities. Type `WAS.clientinfo=all` to pass client information to the backend without tracing of client activities. To deactivate either trace, simply type the trace group name followed by `=off` (without spacing between characters).

About Apache Derby

Use Apache Derby as a test and development database only. Apache Derby must run at a minimal version of v10.3 or Cloudscape v10.1x. The Apache Derby package that is bundled with the application server is backed by full IBM Quality Assurance (QA).

Unlike versions 5.1.60x and earlier, Apache Derby is a pure Java database server. The Apache Derby code base, which the open source community calls Derby, is a product of the Apache Software Foundation (ASF) open source relational database project. Apache Derby includes the Derby base code without any modification to the underlying source code. You can investigate more incompatibilities about Derby code at the Apache Derby web site.

Note: Earlier versions of Apache Derby cannot conduct two phase-commit transactions over the Network Server framework, but later versions of the Derby Client JDBC driver provides Apache Derby with support for XA transactions. Only the Network Server framework provides support for multiple Java virtual machines (JVMs), such as application servers, to access Apache Derby.

Apache Derby is equipped with the following .bat/sh tools:
- sysinfo: displays database version information
- ij: manipulates the database instances

  Note: Use ij as an alternative for the Cloudscape cview tool, which does not exist in Derby.
  When you run the ij tool, surround the dbname by double quotation marks (″ ″) if it includes the full path name; for example:

  ```
  ij> connect '"c:\temp;create=true"'
  ```

  This is ' ″ ″ ' without spaces.
- dblook: dumps DDL information
- networkServerControl: controls the networkServer process (can be used for functions such as ping and trace)
- startNetworkServer: starts the networkServer process
- stopNetworkServer: stops the networkServer process

Verifying the Cloudscape automatic migration

Version 7.0 of the application server requires Cloudscape or Apache Derby to run at a minimal version of v10.1.x. During the application server upgrade to version 7.0, the migration tool automatically upgrades the database instances that are accessed through the embedded framework by some internal components, such as the UDDI registry. The tool also attempts to upgrade Cloudscape or Derby instances that your applications access through the embedded framework. You must verify the migration results for these backend databases.

Before you begin

Do not use Apache Derby or Cloudscape as a production database. Use it for development and test purposes only.

The migration tool attempts to upgrade Cloudscape database instances that are accessed through the embedded framework only. You must manually upgrade Cloudscape instances that transact with application servers on the Network Server framework. See the “Upgrading Cloudscape manually” on page 1542 topic. This requirement eliminates the risk of corrupting third party applications that use the Network Server framework to access the same database instances as WebSphere Application Server.

Other applications can access Apache Derby or Cloudscape on Network Server because the framework provides the database with a foundation of connectivity software; the embedded framework does not. Derby Network Server or Cloudscape Network Server can transact with multiple Java Virtual Machines (JVM) or application servers concurrently, whereas Cloudscape or Derby on the embedded framework works with only a single JVM. Clustered or coexistence implementations of Application Server require Network Server. For more information, consult the IBM Cloudscape Information Center. Find the link in the following IBM Suggests section.

About this task

For database instances that your applications access through the embedded framework, the automatic migration can succeed completely, fail completely, or succeed with warnings. A migration that produces warning messages does create an Apache Derby or Cloudscape database with your data, but does not migrate all of your configured logic and other settings, such as:
- keys

- checks
- views
- triggers
- aliases
- stored procedures

To distinguish between a partially and a completely successful migration, you must verify the auto-migration results by checking both the general post-upgrade log and the individual database logs. Performing these tasks gives you vital diagnostic data to troubleshoot the partially migrated databases as well as those that fail auto-migration completely. Ultimately, you migrate these databases through a manual process.

1. Open the post-upgrade log of each new profile for the application server. The path name of the log is *app_server_root*/profiles/*profileName*/logs/WASPostUpgrade.*timestamp*.log.
2. Examine the post-upgrade log for database error messages. These exceptions indicate database migration failures. The following lines are an example of post-upgrade log content, in which the database error code is `DSRA7600E`. The migration tool references all database exceptions with the prefix `DSRA`.

   ```
   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/cloudscape
   /db2j.properties.

   MIGR0344I: Processing configuration file /opt/WebSphere51/AppServer/config/cells
   /migr06/applications/MyBankApp.ear/deployments/MyBankApp/deployment.xml.

   DSRA7600E: Cloudscape migration of database instance /opt/WebSphere61/Express
   /profiles/default/databases/_opt_WebSphere51_AppServer_bin_DefaultDB failed,
   reason: java.sql.SQLException: Failure creating target db

   MIGR0430W: Cloudscape Database /fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB failed to migrate <new database name>
   ```

   Note: Call IBM WebSphere Application Server Support if you see a migration failure message for a Cloudscape instance that is accessed by a WebSphere internal component (that is, a component that helps comprise WebSphere Application Server rather than one of your applications).
3. Open the individual database migration log that corresponds with each of your backend Cloudscape databases. These logs have the same timestamp as that of the general post-upgrade log. The logs display more detail about errors that are listed in the general post-upgrade log, as well as expose errors that are not documented by the general log.

   The path name of each database log is *app_server_root*/profiles/*profileName*/logs/*myFulldbPathName*_migrationLog*timestamp*.log.
4. Examine each database migration log for errors. For a completely successful migration, the log displays a message that is similar to the following text:

   ```
   MIGR0429I: Cloudscape Database F:\temp\51BaseXExpress\PostUpgrade50BaseFVTTest2\testRun
   \pre\websphere_backup\bin\DefaultDB was successfully migrated. See log C:\WebSphere61
   \Express\profiles\default\logs\DefaultDB_migrationLogSun-Dec-18-13.31.40-CST-2005.log
   ```

   Otherwise, the log displays error messages in the format of the following example:

   ```
   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB>

   connecting to source db <jdbc:db2j:/fvt/temp/51BaseXExpress/PostUpgrade50BaseFVTTest9
   /testRun/pre/websphere_backup/bin/DefaultDB> took 0.26 seconds

   creating target db <jdbc:derby:/opt/WebSphere61/Express/profiles/default/databases
   /_opt_WebSphere51_AppServer_bin_DefaultDB>
   ```

```
 ERROR: An error occurred during migration. See debug.log for more details.

shutting down databases

shutting down databases took   0.055 seconds
```

5. For more data about a migration error, consult the debug log that corresponds with the database migration log. The WebSphere Application Server migration utility triggers a *debug migration trace* by default; this trace function generates the database debug logs. The full path name of a debug log is app_server_root/profiles/*profileName*/logs/*myFulldbPathName*_migrationDebug*timestamp*.log.

   The following lines are a sample of debug text. The lines display detailed exception data for the error that is referenced in the previous sample of database migration log data.

```
java.sql.SQLException: Database_opt_WebSphere51_AppServer_bin_DefaultDB already exists. Aborting migration
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
 at com.ibm.db2j.tools.migration.MigrateFrom51Impl.doMigrate(Unknown Source)
 at com.ibm.db2j.tools.MigrateFrom51.doMigrate(Unknown Source)
 at com.ibm.ws.adapter.migration.CloudscapeMigrationUtility.migr
```

Results

- The migration utility for the application server changes your Apache Derby or Cloudscape JDBC configurations whether or not it successfully migrates the database instances that are accessed by your applications. The tool changes the class paths for the Derby or Cloudscape JDBC provider, data source implementation classes, and data source helper classes. The following table depicts these changes:

Table 14. New class information

Class type	Old value	New value
JDBC provider class path	`${CLOUDSCAPE_JDBC_DRIVER_PATH}/db2j.jar`	`${DERBY_JDBC_DRIVER_PATH}/derby.jar` • Where `DERBY_JDBC_DRIVER_PATH` is the WebSphere environment variable that defines your Cloudscape JDBC provider • Where `derby.jar` is the base name of the JDBC driver class file (In your environment, reference the JDBC driver class file by the full path name.)
Data source implementation class: Connection pool	`com.ibm.db2j.jdbc.DB2jConnectionPool DataSource`	`org.apache.derby.jdbc.EmbeddedConnection PoolDataSource`
Data source implementation class: XA	`com.ibm.db2j.jdbc.DB2jXADataSource`	`org.apache.derby.jdbc.EmbeddedXADataSource`
Data source helper class	`com.ibm.websphere.rsadapter.Cloudscape DataStoreHelper`	`com.ibm.websphere.rsadapter.Derby DataStoreHelper`

  Additionally, the db2j.properties file changes:
  - The name app_server_root/cloudscape/dbj.properties changes to app_server_root/derby/derby.properties
  - Within the file, property names change from db2j.drda.* to derby.drda.*
- A partial or a completely successful database migration changes the location and name of the database according to the following example:
  - **Old database name:** `c:\temp\`*mydb*
  - **New database name:** The new name includes a hash code that combines the entire path name of the old database and the migration time stamp. The new name also includes the old database name and time stamp exactly. For example:

    app_server_root\profiles\profile_name\databases*my_database*_hashCode_timestamp

  Note: For both partial and failed migrations, the log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

What to do next

If you experience a partial migration, attempt to troubleshoot the Cloudscape or Derby database only if you have expert knowledge of these database types. Otherwise, delete the new database. Perform the manual

migration procedure on the original database, just as you do for each database that completely fails auto-migration. Consult “Upgrading Cloudscape manually” for instructions.

After a successful database migration, reboot the database and compress tables to improve performance. See the Apache Derby documentation for instructions.

Upgrading Cloudscape manually

During the upgrade of your application server, the migration tool attempts to upgrade instances of Cloudscape that are accessed through the embedded framework only. The automatic upgrade excludes Cloudscape instances that transact with applications through the Network Server framework. This exclusion eliminates the risk of corrupting third party applications that access the same database instances as the application server. You must manually upgrade database instances that are accessed through the Network Server framework. Do the same for databases that fail the automatic migration.

Before you begin

Do not use Apache Derby Version 10.1.x or Cloudscape v10.1.x as a production database. Use them for development and test purposes only.

For instances of Cloudscape that are accessed through the embedded framework, determine which instances completely failed the automatic upgrade process and which ones were only partially upgraded. The topic on verifying the Cloudscape automatic migration documents how to uncover database errors and diagnostic data from various migration logs. The log messages contain the exact old and new database path names that you must use to run the manual migration. Note these new path names precisely.

To minimize the risk of migration errors for databases that were only partially upgraded during the automatic migration process, delete the new database. Troubleshoot the original database according to the log diagnostic data, then perform manual migration on the original database.

About this task

The following section consists of steps to migrate Cloudscape instances that are accessed through both the embedded framework as well as the Network Server framework. Steps that apply only to the Cloudscape Network Server framework are marked accordingly. As a migration best practice, ensure that your user ID has one of the following authorities:

- Administrator of the application server that accesses the Cloudscape instance
- A umask that can access the database instance

Otherwise, you might see runtime errors about the database instance being read-only.

1. **Network Server framework only:** Ensure that every client of the Cloudscape database can support Cloudscape v10.1.x or Apache Derby. Application server clients of the database must run versions 6.02.x or later of the application server.
2. **Network Server framework only:** Take the database offline. No clients can access it during the migration process.
3. Examine a sample Cloudscape migration script that the application server provides, either db2jmigrate.bat or db2jmigrate.sh. The path of both scripts is *app_server_root*\derby\bin\embedded. You can modify the script according to the requirements of your environment. Consult the Cloudscape migration document for information about options that you can use with the script. For example, you can use the following option to specify the DDL file for the new database:

   `-DB2j.migrate.ddlFile=`*`filename`*
4. To generate database debug logs when you run the migration script, ensure that the debug migration trace is active. By default, this trace function is enabled. Reactivate the debug trace if it is disabled.
   a. To set the trace options in the administrative console, click **Troubleshooting** > **Logging and Tracing** in the console navigation tree.

b. Select the application server name.
c. Click **Change Log Level Details**.
d. Optional: If **All Components** has been enabled, you might want to turn it off, and then enable specific components.
e. Optional: Select a component or group name. For more information see the topic on log level settings. If the selected server is not running, you will not be able to see individual component in graphic mode.
f. Enter a trace string in the trace string box. In this case, enter one of the following:
   - all traces*=all
   - com.ibm.ws.migration.WASUpgrade=all

   For more information on tracing read the topic on working with trace.
g. Select **Apply**, then **OK**.

5. Specify your old database name and the full post-migration path of the new database name when you run the script. For example: `E:\WebSphere\AppServer\derby\bin\embedded>db2jMigrate.bat` *myOldDB myNewDB* The logs from the automatic migration provide the exact path names to specify for both the old database and the target database. You must use this target database name to specify the new database, because your migrated Cloudscape data sources (updated by the WebSphere Application Server migration utilities) now point to the target database name. The following sample text demonstrates how log messages display target database names:

   ```
   DSRA7600E: Cloudscape migration of database instance C:\temp\migration2\profiles\AppSrv01\
   installedApps\ghongellNode01Cell\DynamicQuery.ear\EmployeeFinderDB to new database instance
   C:\WebSphere\AppServer\profiles\AppSrv01\databases\C__WAS602_AppServer_profiles_AppSrv01_
   installedApps_ghongellNode01Cell_DynamicQuery.ear_EmployeeFinderDB failed,
   reason: java.sql.SQLException: Failure creating target db
   ```

   For instances of Cloudscape that are accessed through the Network Server framework, input any name that you want for the new database. Remember to modify your existing data sources to point to the new database name.

6. When the migration process ends, examine the database migration log to verify the results. The path name of each database migration log is *app_server_root*/logs/derby/*myFulldbPathName*_migrationLog.log.

   For a successful migration, the database migration log displays a message that is similar to the following text:

   ```
   Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
   Migration Completed Successfully
   E:\WebSphere\AppServer\derby\bin\embedded>
   ```

   Otherwise, the log displays error messages in the format of the following example:

   ```
   Check E:\WebSphere\AppServer\derby\myOldDB_migrationLog.log for progress
   ERROR: An error occurred during migration. See debug.log for more details.
   ERROR XMG02: Failure creating target db
   java.sql.SQLException: Failure creating target db
           at com.ibm.db2j.tools.migration.MigrationState.getCurrSQLException(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.handleException(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.go(Unknown Source)
           at com.ibm.db2j.tools.migration.MigrateFrom51Impl.main(Unknown Source)
           at com.ibm.db2j.tools.MigrateFrom51.main(Unknown Source)
   ```

   ...

7. For more data about a migration error, consult the debug log that corresponds with the database migration log. The full path name of a debug log file is *app_server_root*/logs/derby/*myFulldbPathName*_migrationDebug.log.

   The following lines are a sample of debug text.

```
sourceDBURL=jdbc:db2j:E:\WebSphere\myOldDB
 newDBURL=jdbc:derby:e:\tempo\myNewDB
 ddlOnly=false
connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB>
connecting to source db <jdbc:db2j:E:\WebSphere\myOldDB> took   0.611 seconds
creating target db <jdbc:derby:e:\tempo\myNewDB>
creating target db <jdbc:derby:e:\tempo\myNewDB> took   6.589 seconds
initializing source db data structures
initializing source db data structures took   0.151 seconds
recording DDL to create db <E:\WebSphere\myOldDB>
recording DDL to create db <E:\WebSphere\myOldDB> took   5.808 seconds
```

Results

As indicated in the previous steps, the database migration log displays either a `Migration Completed Successfully` message, or a message containing migration failure exceptions.

What to do next

- For databases that fail migration, troubleshoot according to the logged error data. Then rerun the migration script.
- To access successfully upgraded databases through the embedded framework, modify your data sources to point to the new database names.
- To access successfully upgraded databases through the Network Server framework, you can use either the DB2 Universal JDBC driver or the Derby Client JDBC driver.
  - If you want your existing JDBC configurations to continue to use the DB2 Universal JDBC driver, modify your data sources to point to the new database names.
  - If you want to use the Derby Client JDBC driver, which can support XA data sources, modify your JDBC providers to use the new Derby Client JDBC driver class and the new data source implementation classes. Then reconfigure every existing data source to use the correct Derby data source helper class, and to point to the new database name.

    Consult the article in the information center on vendor-specific data sources minimum required settings for all of the new class names.

Database performance tuning

Database performance tuning can dramatically affect the throughput of your application. For example, if your application requires high concurrency (multiple, simultaneous interactions with backend data), an improperly tuned database can result in a bottleneck. Database access threads accumulate in a backlog when the database is not configured to accept a sufficient number of incoming requests.

Because WebSphere Application Server supports the integration of many different database products, each one with unique tuning configurations, consult your database vendor documentation for comprehensive tuning information. This information center provides introductory material on for your convenience:

- “DB2 Universal Database performance tips”

DB2 Universal Database performance tips

You can easily adjust your system QSQSRVR prestart job settings to optimize the process of acquiring connections from DB2 Universal Database for i5/OS.

On the i5/OS platform, QSQSRVR jobs process Java Database Connectivity (JDBC) tasks. By default, five QSQSRVR jobs are initially active. When fewer than two QSQSRVR jobs are unused, the i5/OS system creates two more jobs. If you increase the active job values, an application that establishes a large number of database connections over a short period of time might create connections more quickly. Increase the values for the initial number of jobs, threshold, and additional number of jobs by running this command on an i5/OS command line:

```
CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQSRVR)
```

Do not start more QSQSRVR jobs than your application requires. Active QSQSRVR jobs require some overhead, even if these jobs are not used.

Managing resources through JCA lifecycle management operations

You can manage the run-time status of your data source and connection factory resources to perform some data access administrative tasks without restarting the application server. This topic outlines the process for managing those resources through the administrative console.

Before you begin

When you manage the run-time status of connection factories or data sources, you are applying Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) lifecycle management operations to the MBeans that are associated with these resources. The management operations are PAUSE and RESUME. Pausing an MBean halts outbound communication to the backend, such as a database. This action affects all applications that use the corresponding connection factory or data source on the same server.

About this task

With these management operations, you can perform some administrative tasks dynamically, without restarting your application server:

- Respond to a security threat, and prevent new connection requests from reaching the backend
- Perform maintenance on the backend
- Apply configuration changes to non-required properties of the connection factory or data source, such as turning JDBC trace on or off, or modifying preferences for collecting client information

1. Navigate to the administrative console page that corresponds to the resource type that you want to manage.
   - For connection factories, use either of the following paths:
     - **Resources** > **Resource Adapters** > **J2C connection factories**
     - **Resources** > **Resource Adapters** > **Resource Adapters** > *resource_adapter* > **J2C connection factories**
   - For data sources, use either of the following paths:
     - **Resources** > **JDBC** > **Data sources**
     - **Resources** > **JDBC** > **JDBC providers** > *JDBC_provider* > **Data sources**
2. Select the connection factory or data source configurations that you want to manage, and click **Manage state**. The administrative console now displays the JCA lifecycle management page, which contains a table that depicts the full scope configuration of your previous selection.

   The table is comprised of three columns:
   - **JNDI name:** The Java Naming and Directory Interface (JNDI) name of the connection factory or data source configuration.
   - **Running object scope:** The server that is running the connection factory or data source MBean.
   - **Status:** The status of the connection factory or data source MBean.
3. Select the rows that represent each invocation of the resource, per running server, that you want to manage. Be aware that when you click your management operation, WebSphere Application Server applies it to every resource object in your selection.

   Note: If the MBean status of a row has a value of NOT_ACCESSED, you cannot apply JCA lifecycle management operations to that MBean. The NOT_ACCESSED state indicates that the MBean exists on the specified server, but no applications performed a JNDI namespace lookup on the corresponding connection factory or data source.

4. Click **Pause** or **Resume**. The status column of the table changes to reflect the new state of the MBean.

JCA lifecycle management

Use this page to perform JCA lifecycle management operations on data source and connection factory MBeans. With these management operations, you can control the runtime status of the corresponding data source and connection factory resources.

You can view this administrative console page in different locations, depending on whether you want to manage data sources or J2C connection factories. For example:

- For connection factories: Click **Resources** → **Resource adapters** → **J2C connection factories**. Select the connection factory configurations that you want to manage, and click **Manage state**.
- For data sources, click **Resources** → **JDBC** → **Data sources**. Select the data source configurations that you want to manage, and click **Manage state**.

Guidelines for using this administrative console page:

- The table displays a list of MBeans that correspond to the data sources or connection factories in your selection from the previous console page. These MBeans are compatible with Version 6.0.2 and later of the application server.
- You can only perform JCA lifecycle management actions on MBeans that are in the active state. In this context, an MBean is considered active when an application performs a Java Naming and Directory Interface (JNDI) name space lookup on the corresponding data source or connection factory resource.
- Pausing an MBean halts outbound communication to the backend, such as a database. This action affects all applications that use the resource on the selected server.

Pause:

Specifies to pause the MBean that is selected. Pausing an MBean halts outbound communication to the back end resource, and this action will affect all applications that use the resource on the selected server.

Resume:

Specifies to resume the MBean that is selected. Resuming an MBean will enable outbound communication to the back end resource, and this action will affect all applications that use the resource on the selected server.

Purge:

Specifies to purge the contents of the connection pool for the data source or connection factory that is specified. Purging the pool will not affect ongoing transactions.

Name (JNDI name):

The name of the connection factory or data source configuration, followed by the Java Naming and Directory Interface (JNDI) name in parenthesis.

Running object scope:

The server that is running the connection factory or data source MBean.

Status:

The state of the connection factory or data source MBean.

Possible values:

State	Indications
ACTIVE	• The resource that corresponds with the MBean is ready to provide an application with connections to a backend. • An application performed a JNDI namespace lookup on this resource. You can apply the JCA lifecycle management operation of PAUSE to an MBean in this state.
PAUSED	• All outbound communication to the backend through the corresponding resource is stopped, as a result of a JCA lifecycle management operation that was applied previously to the MBean. • An application performed a JNDI namespace lookup on the resource. You can apply the JCA lifecycle management operation of RESUME to an MBean in this state.
NOT_ACCESSED	• The MBean exists on the specified server, but no applications performed a JNDI name space lookup on the corresponding connection factory or data source. You cannot apply JCA lifecycle management operations to an MBean in this state.

Data access problems

WebSphere Application Server diagnostic tools provide services to help troubleshoot database connection problems. Additionally, the IBM Web site provides flexible searching capabilities for finding documented solutions to database-specific connection problems.

The following steps help you quickly isolate connectivity problems.

1. Browse the log files of the application server for clues.

   See Viewing JVM logs. By default, these files are *app_server_root*/*server_name*/SystemErr.log and SystemOut.log.
2. Browse the Helper Class property of the data source to verify that it is correct and that it is on the WebSphere Application Server class path. Mysterious errors or behavior might result from a missing or misnamed Helper Class name. If WebSphere Application Server cannot load the specified class, it uses a default helper class that might not function correctly with your database manager.
3. Verify that the Java Naming and Directory Interface (JNDI) name of the data source matches the name used by the client attempting to access it. If error messages indicate that the problem might be naming-related, such as referring to the **name server** or **naming service**, or including error IDs beginning with **NMSV**, look at the Naming related problems and Troubleshooting the naming service component topics.
4. Enable tracing for the resource adapter using the trace specification, RRA=all=enabled. Follow the instructions for dumping and browsing the trace output, to narrow the origin of the problem.

For a comprehensive list of database-specific troubleshooting tips, see the WebSphere Application Server product support page. (Find the link at the end of this article.) In the Search Support field, type a database vendor name among your search terms. Select **Solve a problem**, then click **Search**.

Remember that you can always find Support references in the Troubleshooting help from IBM article of this information center.

Currently this information center provides a limited number of troubleshooting tips for the following databases:
- Oracle
- DB2
- SQL Server
- Apache Derby
- Sybase
- General data access problems

General data access problems
- An exception ″IllegalConnectionUseException″ occurs
- WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred.
- ConnectionWaitTimeoutException.
- com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be found. SQLSTATE=42705
- java.sql.SQLException: java.lang.UnsatisfiedLinkError:
- ″J2CA0030E: Method enlist caught java.lang.IllegalStateException″ wrapped in error ″WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred″ when attempting to execute a transaction.
- java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation
- ″J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*″ when attempting a database operation
- An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms
- Container-managed persistence (CMP) cannot successfully obtain the database access function as defined.

IllegalConnectionUseException

This error can occur because a connection obtained from a WAS40DataSource is being used on more than one thread. This usage violates the J2EE 1.3 programming model, and an exception generates when it is detected on the server. This problem occurs for users accessing a data source through servlets or bean-managed persistence (BMP) enterprise beans.

To confirm this problem, examine the code for connection sharing. Code can inadvertently cause sharing by not following the programming model recommendations, for example by storing a connection in an instance variable in a servlet, which can cause use of the connection on multiple threads at the same time.

WTRN0062E: An illegal attempt to enlist multiple one phase capable resources has occurred

This error can occur because:
- An attempt was made to share a single-phase connection, when each **getConnection** method has different connection properties; such as the AccessIntent. This attempt causes a non-shareable connection to be created.
- An attempt was made to have more than one unshareable connection participate in a global transaction, when the data source is not an XA resource.
- An attempt was made to have a one-phase resource participate in a global transaction while an XA resource or another one-phase resource already participated in this global transaction.
  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is unshareable, and the data source is not configured to support two-phase commit transactions. It does not support an XAResource. If you do not use a resource-ref, you default to unshareable connections.
  - Within the scope of a global transaction you try to get a connection more than once and at least one of the resource-refs you use specifies that the connection is shareable and the data source is not configured to support two-phase commit transactions. That is, it does not support an XAResource. In

addition, even though you specify that connections are shareable, each getConnection request is made with different connection properties (such as IsolationLevel or AccessIntent). In this case, the connections are not shareable, and multiple connections are handed back.
  - Multiple components (servlets, session beans, BMP entity beans, or CMP entity beans) are accessed within a global transaction. All use the same data source, all specify shareable connections on their resource-refs, and you expect them to all share the same connection. If the properties are different, you get multiple connections. AccessIntent settings on CMP beans change their properties. To share a connection, the AccessIntent setting must be the same. For more information about CMP beans sharing a connection with non-CMP components, see the *Data access application programming interface support* and *Example: Accessing data using IBM extended APIs to share connections between container-managed and bean-managed persistence beans* topics in the DataAccess section of the information center.

To correct this error:
- Check what your client code passes in with its getConnection requests, to ensure they are consistent with each other.
- Check the connection sharing scope from the resource binding, using an assembly tool.
  - If you are running an unshareable connection scope, verify that your data source is an XA data source.
  - If you are running a shareable connection scope, verify that all connection properties, including AccessIntent, are sharable.
- Check the JDBC provider implementation class from the Manage JDBC resource panel of the administrative console to ensure that it is a class that supports XA-type transactions.

ConnectionWaitTimeoutException accessing a data source or resource adapter

If your application receives exceptions like a com.ibm.websphere.ce.cm.ConnectionWaitTimeoutException or com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException when attempting to access a WebSphere Application Server data source or JCA-compliant resource adapter, respectively, some possible causes are:
- The maximum number of connections for a given pool is set too low. The demand for concurrent use of connections is greater then the configured maximum value for the connection pool. One indication that this situation is the problem is that you receive these exceptions regularly, but your CPU utilization is not high. This exception indicates that there are too few connections available to keep the threads in the server busy.
- Connection Wait Time is set too low. Current demand for connections is high enough such that sometimes there is not an available connection for short periods of time. If your connection wait timeout value is too low, you might timeout shortly before a user returns a connection back to the pool. Adjusting the connection wait time can give you some relief. One indication of this problem is that you use close to the maximum number of connections for an extended period and receiving this error regularly.
- You are not closing some connections or you are returning connections back to the pool at a very slow rate. This situation can happen when using unshareable connections, when you forget to close them, or you close them long after you are finished using them, keeping the connection from returning to the pool for reuse. The pool soon becomes empty and all applications get ConnectionWaitTimeoutExceptions. One indication of this problem is you run out of connections in the connection pool and you receive this error on most requests.
- You are driving more load than the server or backend system have resources to handle. In this case you must determine which resources you need more of and upgrade configurations or hardware to address the need. One indication of this problem is that the application or database server CPU is nearly 100% busy.

To correct these problems, either:
- Modify an application to use fewer connections
- Properly close the connections.

- Change the pool settings of MaxConnections or ConnnectionWaitTimeout.
- Adjust resources and their configurations.

com.ibm.websphere.ce.cm.StaleConnectionException: [IBM][CLI Driver] SQL1013N The database alias name or database name ″NULL″ could not be found. SQLSTATE=42705

This error occurs when a data source is defined but the **databaseName** attribute and the corresponding value are not added to the custom properties panel.

To add the **databaseName** property:
1. Click **Resources>Manage JDBC Providers** link in the administrative console.
2. Select the JDBC provider that supports the problem data source.
3. Select **Data Sources** and then select the problem data source.
4. Under **Additional properties** click **Custom Properties**.
5. Select the **databaseName** property, or add one if it does not exist, and enter the actual database name as the value.
6. Click **Apply** or **OK**, and then click **Save** from the action bar.
7. Access the data source again.

java.sql.SQLException: java.lang.UnsatisfiedLinkError:

This error indicates that the directory containing the binary libraries which support a database are not included in the LIBPATH environment variable for the environment in which the WebSphere Application Server starts.

The path containing the DBM vendor libraries vary by dbm. One way to find them is by scanning for the missing library specified in the error message. Then you can correct the LIBPATH variable to include the missing directory, either in the `.profile` of the account from which WebSphere Application Server is executed, or by adding a statement in a `.sh` file which then executes the `startServer` program.

″J2CA0030E: Method enlist caught java.lang.IllegalStateException″ wrapped in error ″WTRN0063E: An illegal attempt to enlist a one phase capable resource with existing two phase capable resources has occurred″ when attempting to execute a transaction.

This error can occur when last participant support is missing or disabled. last participant support allows a one-phase capable resource and a two-phase capable resource to enlist within the same transaction.

Last participant support is only available if the following are true:
- WebSphere Application Server Programming Model Extensions (PME) is installed. PME is included in the Application Server Integration Server product.
- The Additional Integration Server Extensions option is enabled when PME is installed. If you perform a typical installation, this function is enabled by default. If you perform a custom installation, you have the option to disable this function, which disables last participant support.
- The application enlisting the one-phase resource is deployed with the **Accept heuristic hazard** option enabled. This deployment is done with an assembly tool.

java.lang.UnsatisfiedLinkError:xaConnect exception when attempting a database operation

This problem has two main causes:
- The most common cause is that the jdbc driver which supports connectivity to the database is missing, or is not the correct version, or that native libraries which support the driver are on the system's path.

– To resolve this problem on a Windows platform, verify that the JDBC driver jar file is on the system PATH environment variable:
  - If you are using DB2,verify that at least the DB2 client product has been installed on the WebSphere host
    • On DB2 version 7.2 or earlier, the file where the client product is installed on the WebSphere Application Server is db2java.zip. Verify that the usejdbc2.bat program has been executed after the database install and after any upgrade to the database product.
    • On DB2 version 8.1 or later, use the DB2 Universal JDBC Provider Driver when defining a JDBC provider in WebSphere Application Server. The driver file is db2jcc.jar. If you use the type 2 (default) option, verify that at least the DB2 client product is installed on the WebSphere Application Server host. If you specify the type 4 option, the DB2 client does not need to be installed, but the file db2jcc.jar still must be present.

      When specifying the location of the driver file, it is recommended to that you specify the path and file name of the target DB2 installation, rather than simply copying the file to a local directory, if possible. Otherwise, you may be exposed to problems if the target DB2 installation is upgraded and the driver used by WebSphere Application Server is not.
– On operating systems such as AIX or Linux, ensure that any native libraries required to support the database client of your database product are specified in the LD_LIBRARY_PATH environment variable in the profile of the account under which WebSphere Application Server executes.

  If you are using DB2 The native library is libdb2jdbc.so. The best way to ensure that this library is accessed correctly by WebSphere is to call the db2profile script supplied with DB2 from the .profile script of the account (such as ″root″) under which WebSphere runs.
  - If you are using DB2 version 7.2 or earlier, ensure that the usejdbc2,script provided with DB2 is called from the profile of the account under which WebSphere Application server is launched.
  - If you are using DB2 version 8.1 or later, see the previous instructions for the Windows operating system.

• If the database manager is DB2, you may have chosen the option to create a 64-bit instance. Sometimes a 64-bit configuration is not supported. If this has happened, remove the database instance and create a new one with the default 32-bit setting.

  If you are using the Universal JDBC T2 driver, WebSphere Application Server does support interaction with a DB2 UDB 64-bit server, but it must be through a DB2 UDB 32-bit client. The WebSphere Application Server environment (CLASSPATH and so on) must use the 32-bit client code to ensure correct function.

  With a Universal JDBC T4 driver, you do not need the 32-bit DB2 client. You need only configure the CLASSPATH to include db2jcc.jar and its license files in the WebSphere Application Server environment.

  Note: For general help in configuring JDBC drivers and data sources in WebSphere Application Server, see the topic Accessing data from applications.

″J2CA0114W: No container-managed authentication alias found for connection factory or datasource *datasource*″ when attempting a database operation

This error might occur in the SystemOut.log file when you run an application to access a data source after creating the data source using JACL script.

The error message occurs because the JACL script did not set container-managed authentication alias for CMP connection factory. The JACL is missing the following line:

```
$AdminConfig create MappingModule $cmpConnectorFactory  "{mappingConfigAlias
DefaultPrincipalMapping} {authDataAlias $authDataAlias}
```

To correct this problem, add the missing line to the JACL script and run the script again. See “Example: Creating a JDBC provider and data source using Java Management Extensions API and the scripting tool” on page 1486 for a sample JACL script.

An error is thrown if you use the ws_ant command to perform the database customization for Structured Query Language in Java on HP platforms

If you use the `ws_ant` command to perform the database customization for Structured Query Language in Java (SQLJ) on HP platforms, you can receive an error similar to the following:

```
[java] [ibm][db2][jcc][sqlj]
[java] [ibm][db2][jcc][sqlj] Begin Customization
[java] [ibm][db2][jcc][sqlj] encoding not supported!!
```

The cause of this error might be that your databases were created using the HP default character set. The Java Common Client (JCC) driver depends on the software development kit (SDK) to perform the codepage conversions. The SDK shipped with this product, however, does not support the HP default codepage.

You need to set your LANG to the ISO locale before creating the databases. It should be similar to the following:

```
export LANG=en_US.iso88591
```

Refer to the IBM support site for Information Management software to access the latest technotes for DB2.

Container-managed persistence (CMP) cannot successfully obtain the database access function as defined.

When WebSphere Application Server is caching certain generated code that is accessed in the database on the connection factory, and if any changes in the Java archive (JAR) file require regeneration of the database access, the changes are not effective until you stop and restart the server.

Examples of when this failure might occur include:

- Adding an enterprise bean custom finder method; a NullPointerException exception is created.
- Updating an enterprise bean custom finder method; the new SQL statement does not run.
- Changing schema mapping; the new SQL statement does not run.

In summary, if you add or update an enterprise bean that contains a custom finder method, you must stop and then restart the server.

Data access problems - Oracle data source

This article provides troubleshooting tips for accessing Oracle data sources.

What kind of error do you see when you try to access your Oracle-based data source?

• “Transaction recovery failure (for XA data sources)” on page 1555

An invalid Oracle URL is specified

This error might be caused by an incorrectly specified URL on the URL property of the target data source.

Examine the URL property for the data source object in the administrative console. For the 8i OCI driver, verify that **oci8** is used in the URL. For the 9i OCI driver, you can use either **oci8** or **oci**.

Examples of Oracle URLs:
- For the thin driver: `jdbc:oracle:thin:@hostname.rchland.ibm.com:1521:IBM`
- For the thick (OCI) driver: `jdbc:oracle:oci8:@tnsname1`

″DSRA0080E: An exception was received by the data store adapter. See original exception message: ORA-00600″ when connecting to or using an Oracle data source

A possible reason for this exception is that the version of the Oracle JDBC driver being used is older than the Oracle database. It is possible that more than one version of the Oracle JDBC driver is configured on the WebSphere Application Server.

Examine the version of the JDBC driver. Sometimes you can determine the version by looking at the class path to determine what directory the driver is in.

If you cannot determine the version this way, use the following program to determine the version. Before running the program, set the class path to the location of your JDBC driver files.

```
import java.sql.*;
     import oracle.jdbc.driver.*;
     class JDBCVersion
     {
         public static void main (String args[])
         throws SQLException
         {
             // Load the Oracle JDBC driver
             DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
             // Get a connection to a database
             Connection conn = DriverManager.getConnection
             ("jdbc:oracle:thin:@appaloosa:1521:app1","sys","change_on_install");
             // Create Oracle DatabaseMetaData object
             DatabaseMetaData meta = conn.getMetaData();
             // gets driver info:
             System.out.println("JDBC driver version is " + meta.getDriverVersion());
         }
     }
```

If the driver and the database are at different versions, replace the JDBC driver with the correct version. If multiple drivers are configured, remove any that occur at the incorrect level.

DSRA8100E: Unable to get a {0} from the DataSource. Explanation: See the linkedException for more information.

When using an oracle thin driver, Oracle creates a ″java.sql.SQLException: invalid arguments in call″ error if no user name or password is specified when getting a connection. If you see this error while running WebSphere Application Server, the alias is not set.

To remove the exception, define the alias on the data source.

″Error while trying to retrieve text for error″ error when connecting to an Oracle data source

The most likely cause of this error is that the Oracle 8i OCI driver is being used with an ORACLE_HOME property that is either not set or is set incorrectly.

To correct the error, examine the user profile that WebSphere Application Server is running under to verify that the $ORACLE_HOME environment variable is set correctly.

″java.lang.UnsatisfiedLinkError:″ connecting to an Oracle data source

The environment variable LIBPATH might not be set or is set incorrectly, if your data source creates an **UnsatisfiedLinkError** error, and the full exception indicates that the problem is related to an Oracle module, as in the following examples.

Example of invalid an LIBPATH for the 8i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so:
load ENOENT on shared library(s)
/usr/WebSphere/AppServer/java/jre/bin/libocijdbc8.so libclntsh.a
```

Example of an invalid LIBPATH for the 9i driver:

```
Exception in thread "main" java.lang.UnsatisfiedLinkError:
no ocijdbc9  (libocijdbc9.a or .so) in java.library.path
at java.lang.ClassLoader.loadLibrary(ClassLoader.java(Compiled Code))
at java.lang.Runtime.loadLibrary0(Runtime.java:780)
```

To correct the problem:

1. Examine the user profile under which the WebSphere Application Server is running to verify that the LIBPATH environment variable includes Oracle libraries. Scan for the `lobocijdbc8.so` file to find the right directory.

java.lang.NullPointerException referencing 8i classes, or ″ internal error: oracle.jdbc.oci8. OCIEnv″ connecting to an Oracle data source

The problem might be that the 9i OCI driver is being used on an AIX 32-bit machine, the LIBPATH is set correctly, but the ORACLE_HOME environment variable is not set or is set incorrectly. You can encounter an exception similar to either of the following when your application attempts to connect to an Oracle data source:

Exception example for the java.lang.NullPointerException:

```
Exception in thread "main" java.lang.NullPointerException
at oracle.jdbc.oci8.OCIDBAccess.check_error(OCIDBAccess.java:1743)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:69)
at oracle.jdbc.oci8.OCIDBAccess.logon(OCIDBAccess.java:452)
at oracle.jdbc.driver.OracleConnection. <init>(OracleConnection.java:287)
```

Exception example for the java.sql.SQLException:

```
Exception in thread "main" java.sql.SQLException:
internal error: oracle.jdbc.oci8. OCIEnv@568b1d21
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:184)
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:226)
at oracle.jdbc.oci8.OCIEnv.getEnvHandle(OCIEnv.java:79)
```

To correct the problem, examine the user profile that WebSphere Application Server is running under to verify that it has the `$ORACLE_HOME` environment variable set correctly, and that the `$LIBPATH` includes `$ORACLE_HOME/lib`.

WSVR0016W: Classpath entry for the Oracle JDBC Thin Driver has an invalid variable

This error occurs when there is no environment variable defined for the property, ORACLE_JDBC_DRIVER_PATH.

Verify this problem in the administrative console. Go to **Environment > Manage WebSphere Variables** to verify whether the variable `ORACLE_JDBC_DRIVER_PATH` is defined.

To correct the problem, click **New** and define the variable. For example, name : **ORACLE_JDBC_DRIVER_PATH** , value : **c:\oracle\jdbc\lib**. Use a value that names the directory in your operating system that contains the `ojdbc6.jar` file (or the `ojdbc14_g.jar` file to enable Oracle trace).

Transaction recovery failure (for XA data sources)

Problem

When WebSphere Application Server attempts to recover Oracle database transactions, the transaction service issues the following exception:

```
WTRN0037W: The transaction service encountered an error on an xa_recover operation.
The resource was com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl@1114a62.
The error code was XAER_RMERR. The exception stack trace follows:
javax.transaction.xa.XAException
at oracle.jdbc.xa.OracleXAResource.recover(OracleXAResource.java:726)
at com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl.recover(WSRdbXaResourceImpl.java:954)
at com.ibm.ws.Transaction.JTA.XARminst.recover(XARminst.java:137)
at com.ibm.ws.Transaction.JTA.XARecoveryData.recover(XARecoveryData.java:609)
at com.ibm.ws.Transaction.JTA.PartnerLogTable.recover(PartnerLogTable.java:511)
at com.ibm.ws.Transaction.JTA.RecoveryManager.resync(RecoveryManager.java:1784)
at com.ibm.ws.Transaction.JTA.RecoveryManager.run(RecoveryManager.java:2241)
```

Cause

Oracle requires services such as the WebSphere Application Server transaction service to have special permissions for performing transaction recoveries.

Solution

As user **SYS**, run the following commands on your Oracle server:

```
grant select on pending_trans$ to public;
grant select on dba_2pc_pending to public;
grant select on dba_pending_transactions to public;
grant execute on dbms_system to <user>;
```

User is a user ID in the application server that is authorized to perform transaction recovery for the XA data source. If you have not authorized any user IDs to perform transaction recovery, the application server will use the login alias for the data source as the user ID.

This problem is mentioned under Oracle bug: 3979190. Running the preceding commands solves the problem.

Data access problems - DB2 database

This article provides troubleshooting tips for accessing DB2 databases.

What kind of problem are you having accessing your DB2 database?

- “SQL30082N Attempt to establish connection failed with security reason "17" ("UNSUPPORTED FUNCTION’) SQLSTATE=08001” on page 1557
- “SQLException, with ErrorCode -99,999 and SQLState 58004, with Java ″StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004″, when using WAS40-type data source” on page 1557
- “Error message java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class ″COM.ibm.db2.jdbc.DB2XADataSource″ could not be found. when trying to access a DB2 database” on page 1558
- “CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=5800” on page 1558
- “COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code ″2″. SQLSTATE=40001” on page 1558
- “″COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource″ could not be found for data source ([data-source-name])″” on page 1559
- “ java.sql.SQLException: Failure in loading T2 native library db2jcct2 DSRA0010E: SQL State = null, Error Code = -99,999 ” on page 1560
- Lock contention exception occurs in database when data source implementation type is XA
- “″DSRA8050W: Unable to find the DataStoreHelper class specified″ exception occurs when trying to use a DB2 Universal Datasource in a mixed release cell.” on page 1560
- “Receive "'SYSTEM’ is not a valid authorization ID" message when trying to access DB2 on a Windows machine where WebSphere Application Server is also installed.” on page 1561
- “XAException: XAER_NOTA on XA prepare call in DB2 Universal JDBC Driver type 4 after one phase transaction rollback” on page 1561
- “java.rmi.MarshalException logged for application client due to incompatibility of JDBC driver file versions” on page 1562
- “Database failure triggers problematic -99999 exception for applications that use DB2 Universal Driver type 4” on page 1562
- “Cannot access DB2 on Linux when using the DB2 Universal JDBC Driver” on page 1563
- “Data access problems - DB2 database” on page 1555
- “Illegal conversion occurs on any VARCHAR FOR BIT DATA column in a container-managed persistent bean” on page 1564

SQL0567N ″DB2ADMIN ″ is not a valid authorization ID. SQLSTATE=42602

If you encounter this error when attempting to access a DB2 Universal Database (UDB):
1. Verify that your user name and password in the data source properties page in the administrative console are correct.
2. Ensure that the user ID and password do not contain blank characters before, in between, or after.

SQL0805N Package *package-name* was not found

Possible reasons for these exceptions:
- If the package name is `NULLID.SQLLC300`, see SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002. for the reason.
- You are attempting to use an XA-enabled JDBC driver on a DB2 database that is not XA-ready.

To correct the problem on a DB2 Universal Database (UDB), run this one-time procedure, using the db2cmd interface while connected to the database in question:
1. **DB2 bind @db2ubind.lst blocking all grant public**
2. **DB2 bind @db2cli.lst blocking all grant public**

The `db2ubind.lst` and `db2cli.lst` files are in the `bnd` directory of your DB2 installation root. Run the commands from that directory.

SQL0805N Package ″NULLID.SQLLC300″ was not found. SQLSTATE=51002

This error can occur because:
- The underlying database was dropped and recreated.
- DB2 was ugpraded and its packages are not rebound correctly.

To resolve this problem, rebind the DB2 packages by running the db2cli.lst script found in the bnd directory. For example: db2>@db2cli.lst.

SQL30082N Attempt to establish connection failed with security reason ″17″ (″UNSUPPORTED FUNCTION') SQLSTATE=08001

This error can occur when the security mechanism specified by the client is not valid for this server. Some typical examples:
- The client sent a new password value to a server that does not support the change password function.
- The client sent SERVER_ENCRYPT authentication information to a server that does not support password encryption.
- The client sent a userid, but no password, to a server that does not support authentication by userid only.
- The client has not specified an authentication type, and the server has not responded with a supported type. This can include the server returning multiple types from which the client is unable to choose.

To resolve this problem, ensure that your client and server use the same security mechanism. For example, if this is an error on your data source, verify that you have assigned a user id and password or authentication alias.

SQLException, with ErrorCode -99,999 and SQLState 58004, with Java ″StaleConnectionException: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=58004″, when using WAS40-type data source

An unexpected system failure usually occurs when running in XA mode (two-phase commit). Among the many possible causes are:
- An invalid username or password was provided.
- The database name is incorrect.
- Some DB2 packages are corrupted.

To determine whether you have a user name or password problem, look in the `db2diag.log` file to view the actual error message and SQL code. A message like the following example, with an SQLCODE of -1403, indicates an invalid user ID or password:

```
2002-07-26-14.19.32.762905   Instance:db2inst1   Node:000
PID:9086(java)   Appid:*LOCAL.db2inst1.020726191932
XA DTP Support  sqlxa_open   Probe:101
DIA4701E Database "POLICY2" could not be opened
for distributed transaction processing.
String Title: XA Interface SQLCA  PID:9086 Node:000
SQLCODE = -1403
```

To resolve these problems:
1. Correct your user name and password. If you specify your password on the GUI for the data source, ensure that the user name and password you specify on the bean are correct. The user name and password you specify on the bean overwrite whatever you specify when creating the data source.
2. Use the correct database name.
3. Rebind the packages (in the `bnd` directory) as follows:

```
db2connect to dbname
c:\SQLLIB\bnd>DB2 bind @db2ubind.lst blocking all grant public
c:\SQLLIB\bnd>DB2 bind @db2cli.lst blocking all grant public
```

4. Ensure that the \WebSphere\AppServer\properties\wsj2cdpm.properties file has the right user ID and password.

Error message `java.lang.reflect.InvocationTargetException: com.ibm.ws.exception.WsException: DSRA0023E: The DataSource implementation class "COM.ibm.db2.jdbc.DB2XADataSource" could not be found.` when trying to access a DB2 database

One possible reason for this exception is that a user is attempting to use a JDBC 2.0 DataSource, but DB2 is not JDBC 2.0-enabled. This situation frequently happens with new installations of DB2 because DB2 provides separate drivers for JDBC 1.X and 2.0, with the same physical file name. By default, the JDBC 1.X driver is on the class path.

To confirm this problem:
- On Windows systems, look for the inuse file in the java12 directory in your DB2 installation root. If the file missing, you are using the JDBC 1.x driver.
- On operating systems such as AIX or Linux, check the class path for your data source. If the class path does not point to the `db2java.zip` file in the `java12` directory, you are using the JDBC 1.x driver.

To correct this problem:
- On Windows systems, stop DB2. Run the usejdbc2.bat file from the java12 directory in your DB2 instalation root. Run this file from a command line to verify that it completes successfully.
- On operating systems such as AIX or Linux, change the class path for your data source to point to the `db2java.zip` file in the `java12` directory of your DB2 installation root.

CLI0119E System error. SQLSTATE=58004 - DSRA8100 : Unable to get a XAconnection or DSRA0011E: Exception: COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver] CLI0119E Unexpected system failure. SQLSTATE=5800

If you encounter this error when attempting to access a DB2 Universal Database (UDB) data source:
1. On the data source properties page in the administrative console, verify that the correct database name is specified on the data source.
2. On the custom properties page, check your user name and password custom properties. Verify that they are correct.
3. Ensure the user ID and password do not contain any blank characters, before, in between, or after.
4. Check that the `WAS.policy` file exists for the application, for example, `D:\WebSphere\AppServer\installedApps\markSection.ear\META-INF\was.policy`.
5. View the entire exception listing for an underlying SQL error, and look it up using the DBM vendor message reference.

If you encounter this error while running DB2 on Red Hat Linux, the **max queues system wide** parameter is too low to support DB2 while it acquires the necessary resources to complete the transaction. When this problem exists, the exceptions J2CA0046E and DSRA0010E can precede the exception DSRA8100E.

To correct this problem, edit the /proc/sys/kernal/msgmni file to increase the value of the **max queues system wide** parameter to a value greater than 128.

COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N The current transaction has been rolled back because of a deadlock or timeout. Reason code "2". SQLSTATE=40001

This problem is probably an application-caused DB2 deadlock, particularly if you see an error similar to the following when accessing a DB2 data source:

```
ERROR CODE: -911
COM.ibm.db2.jdbc.DB2Exception: [IBM][CLI Driver][DB2/NT] SQL0911N
The current transaction has been rolled back because of a deadlock or timeout.
Reason code "2".  SQLSTATE=40001
```

To diagnose the problem:
1. Execute these DB2 commands:
   a. `db2 update monitor switches using LOCK ON`
   b. `db2 get snapshot for LOCKS on dbName >`

   The *directory_name*`\lock_snapshot.log` now has the DB2 lock information.
2. Turn off the lock monitor by running: `db2 update monitor switches using LOCK OFF`

To verify that you have a deadlock:
1. Look for an application handle that has a lock-wait status, and then look for the ID of the agent holding lock to verify the ID of the agent.
2. Go to that handle to verify it has a lock-wait status, and the ID of the agent holding the lock for it. If it is the same agent ID as the previous one, then you know that you have a circular lock (deadlock).

To resolve the problem:
1. Examine your application and use a less restrictive isolation level if no concurrency access is needed.
2. Use caution when changing the **accessIntent** value to move to a lower isolation level. This change can result in data integrity problems.
3. For DB2/UDB Version 7.2 and earlier releases, you can set the DB2_RR_TO_RS flag from the DB2 command line window to eliminate unnecessary deadlocks, such as when the accessIntent defined on the bean method is too restrictive, for example, PessimisticUpdate. The DB@_RR_TO_RS setting has two impacts:
   - If RR is your chosen isolation level, it is effectively downgraded to RS.
   - If you choose another isolation level, and the DB2_RR_TO_RS setting is on, a scan skips over rows that are deleted but not committed, even though the row might qualify for the scan. The skipping behavior affects the RR, Read Stability (RS), and Cursor Stability (CS) isolation levels.

   For example, consider the scenario where transaction A deletes the row with column1=10 and transaction B does a scan where column1>8 and column1<12. With DB2_RR_TO_RS off, transaction B waits for transaction A to commit or rollback. If transaction A rolls back, the row with column1=10 is included in the result set of the transaction B query. With DB2_RR_TO_RS on, transaction B does not wait for transaction A to commit or rollback. Transaction B immediately receives query results that do not include the deleted row. Setting DB2_RR_TO_RS effectively changes locking behavior, thus avoiding deadlocks.

″COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource″ could not be found for data source ([data-source-name])″

This error is denoted by message DSRA8040I: Failed to connect to the DataSource.

This error usually occurs when the class path of the DB2 JDBC driver is set correctly to `${DB2_JDBC_DRIVER_PATH}/db2java.zip` but the environment variable DB2_JDBC_DRIVER_PATH is not set.

This error can also occur if you are using DB2 Version 7.1 or 7.2 and you have not yet run *usejdbc2*. This might be the problem if your path is correct but you still receive this error.

To confirm this problem:
1. Go to the **Manage WebSphere Variables** panel.
2. Select **Environment** to verify that there is no entry for the variable DB2_JDBC_DRIVER_PATH.

To correct this problem: Add the variable DB2_JDBC_DRIVER_PATH with **value** equal to the directory path containing the `db2java.zip` file.

java.sql.SQLException: Failure in loading T2 native library db2jcct2 DSRA0010E: SQL State = null, Error Code = -99,999

The *Failure in loading* message indicates one of two things:

- Usually this happens when the machine was not rebooted after installing DB2. Reboot the machine getting the error and try it again.
- The DB2 context is not getting set up correctly for the user running WebSphere Application Server. Source the db2profile file on the machine, and ensure that the environment contains pointers to the DB2 native libraries.

Lock contention exception occurs in database when data source implementation type is XA

Note: Because a lock contention exception can be caused by many factors, consider the following explanation and recommended response as a strategy for eliminating the possible reasons for your lock contention problem.

Symptom	A lock contention exception occurs in a DB2 database that your application accesses through a data source of implementation type XA.
Problem	Your application is trying to access database records that are locked by an XA transaction that is in ended (e) state, but cannot be prepared by the transaction manager.
Description	An XA transaction to DB2 that ends, but cannot be prepared, is in ended (e) state. Because it is *not* considered to be *in doubt*, the transaction manager cannot recover this transaction. DB2 does not return it in the list of in doubt transactions. DB2 also does not roll the transaction back immediately; it waits until all connections to the database are released. During this period of inaction, the transaction continues to hold locks on the database. If your application server does not disconnect all connections from the database to allow rollback, the ended transaction persists in locking the same database records. If your application attempts to access these locked records, a lock contention exception occurs in DB2.
Recommended response	DB2 Version 8.2 is shipped with a sample application that connects to a defined DB2 server and uses the available DB2 APIs to obtain a list of these particular ended transactions. The application offers a configuration setting that enables you to designate an amount of time after which the application rolls these transactions back. Locate the sample application in the `sqllib/samples/db2xamon.c` directory of DB2 Version 8.2 and run it.

″DSRA8050W: Unable to find the DataStoreHelper class specified″ exception occurs when trying to use a DB2 Universal Datasource in a mixed release cell.

This error usually occurs when you are using WebSphere Application Server Version 6.0 or above in conjunction with a previous version and attempt to create a DB2 Universal Datasource on the previous version.

This can happen because the DB2 Universal Datasource was not available on Version 5 and previous versions, but the Version 6 administrative console allows you to build one.

To correct this problem: create the datasource on Version 6.0 or later.

Receive ″'SYSTEM' is not a valid authorization ID″ message when trying to access DB2 on a Windows machine where WebSphere Application Server is also installed.

Symptom	For a WebSphere Application Server on Windows installation that uses DB2 as the backend, you see the following exception in the JVM log:
	`java.sql.SQLException: [IBM][CLI Driver] SQL0567N "SYSTEM" is not a valid authorization ID. SQLSTATE=42602 DSRA0010E: SQL State = 42602, Error Code = -567`
	`at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.throw_SQLException (Unknown Source)`
	`at COM.ibm.db2.jdbc.app.SQLExceptionGenerator.check_return_code (Unknown Source)`
	`at COM.ibm.db2.jdbc.app.DB2Connection.connect(Unknown Source)`
	`at COM.ibm.db2.jdbc.app.DB2Connection.<init>(Unknown Source)`
	`at COM.ibm.db2.jdbc.app.DB2ReusableConnection.<init>(Unknown Source)`
	`at COM.ibm.db2.jdbc.DB2PooledConnection.getConnection(Unknown Source)`
	`at com.ibm.ws.rsadapter.spi.WSRdbDataSource.getConnection (WSRdbDataSource.java:1035)`
	`at com.ibm.ws.rsadapter.spi.WSManagedConnectionFactoryImpl. createManagedConnection(WSManagedConnectionFactoryImpl.java:937)`
	`at com.ibm.ejs.j2c.poolmanager.FreePool. createManagedConnectionWithMCWrapper(FreePool.java:1502)`

Problem	This exception occurs for configurations in which WebSphere Application Server is a client to the DB2 server. The underlying problem is an authorization conflict between WebSphere Application Server on Windows and DB2 that arises when an application attempts to connect to DB2 without providing a user ID and a password.
Description	When a DB2 client and the DB2 database run on the same machine, DB2 allows the client to connect without a user ID and password. The connection is made under the credentials of the user that owns the client process: in this case, the application server JVM. However, if WebSphere Application Server runs as a Windows service, and the ″Log on as″ option is set to ″Local System Account″, the application server JVM is categorized as a subcomponent of a special Windows user called SYSTEM. This user is not allowed to connect to DB2, resulting in the previously shown exception.
Recommended response	You have two options: • Modify the WebSphere Application Server service to use a **Log on as** option of **This account**, and provide an account with permission to connect to DB2. **Or** • Configure your application server to provide credentials on the DB2 connection by using container-managed or component-managed authentication.

XAException: XAER_NOTA on XA prepare call in DB2 Universal JDBC Driver type 4 after one phase transaction rollback

Symptom

For applications that use the DB2 Universal JDBC Driver type 4 XA available with DB2 v8.2, a connection might fail and trigger an `XAER_NOTA XAException` error. The following code block is an example of this exception:

```
J2CA0027E: An exception occurred while invoking prepare
on an XA Resource Adapter from dataSource jdbc/SDOSVT,
within transaction ID {XidImpl: formatId(57415344),
gtrid_length(36), bqual_length(54),
data(000000ff51913982000000010000000296cac5c42fe3c6838631cbaafc8b5a9253b846544
000000ff519139820000000010000000296cac5c42fe3c6838631cbaafc8b5a9253b8465440000000
100000000000000000000000000002)}:
```

```
javax.transaction.xa.XAException: XAER_NOTA
at com.ibm.db2.jcc.a.xb.a(xb.java:1682)
at com.ibm.db2.jcc.a.xb.a(xb.java:841)
at com.ibm.db2.jcc.a.xb.prepare(xb.java:812)
at com.ibm.ws.rsadapter.spi.WSRdbXaResourceImpl.prepare
(WSRdbXaResourceImpl.java:837)
...
```

Problem

If a DB2 Universal JDBC Driver type 4 XA connection is used in a single-phase transaction, such as a local transaction with autocommit set to false, and that single-phase transaction is rolled back, the next use of the connection in a two-phase transaction fails on the prepare call.

A problem in the DB2 Universal JDBC Driver type 4 XA support causes the XA prepare call to fail. This problem does not occur if the single-phase transaction is committed, and it does not occur when using the DB2 Universal JDBC Driver in type 2 mode.

Solution

Upgrade to DB2 Version 8.2 Fix Pack 1, which is equivalent to Version 8.1 Fix Pack 8. The Universal JDBC Driver XA that is available with these releases solves the previously described issue for type 4 mode.

java.rmi.MarshalException logged for application client due to incompatibility of JDBC driver file versions

Symptom

For an application that includes a application client, the following error message is displayed in the client log file of the application server:

```
java.rmi.MarshalException: CORBA MARSHAL
0x4942f89a No; nested exception is:
org.omg.CORBA.MARSHAL: Unable to read value from
underlying bridge : Mismatched serialization
UIDs : Source (Rep.
IDRMI:com.ibm.db2.jcc.c.SqlException:63EEE52211DCD763:82CE0C0DA2B0A000)
= 82CE0C0DA2B0A000 whereas Target (Rep. ID
RMI:com.ibm.db2.jcc.c.SqlException:63EEE52211DCD763:91C6171BC645E41B)
= 91C6171BC645E41B  vmcid: 0x4942f000  minor code:
2202  completed: No
```

Problem

The `db2jcc.jar` files on the application client machine and on your application server are from versions of DB2 that are not compatible with each other, or are not compatible with the version of DB2 that functions as the datastore.

Solution

Check the `db2jcc.jar` files on the application client machine, your application server, and your DB2 server. On the client machine and the application server, install files of the same version that is compatible with the DB2 server.

Database failure triggers problematic -99999 exception for applications that use DB2 Universal Driver type 4

Symptom

If you use the DB2 Universal Driver type 4 for access to DB2 or Cloudscape Network Server, and your database fails, the database server issues a generic -99999 exception in response to every JDBC getConnection request. This exception, which is exemplified in the following code excerpt, can cause unexpected behavior in your applications.

```
java.sql.SQLException: IO Exception opening socket to
 server bs8.rchland.ibm.com on port 1527.
The DB2 Server may be down.DSRA0010E: SQL State = null,
 Error Code = -99,999DSRA0010E: SQL State = null,
Error Code = -99,999
at com.ibm.db2.jcc.b.a.<init>(a.java:125)
at com.ibm.db2.jcc.b.b.a(b.java:1011)
at com.ibm.db2.jcc.c.l.<init>(l.java:197)
at com.ibm.db2.jcc.b.b.<init>(b.java:258)
at com.ibm.db2.jcc.DB2PooledConnection.
 <init>(DB2PooledConnection.java:44)
at com.ibm.db2.jcc.DB2ConnectionPoolDataSource.getPooledConnectionX
 (DB2ConnectionPoolDataSource.java:80)
at com.ibm.db2.jcc.DB2ConnectionPoolDataSource.getPooledConnection
 (DB2ConnectionPoolDataSource.java:45)
at com.ibm.ws.rsadapter.DSConfigurationHelper$1.run
 (DSConfigurationHelper.java:945)
```

Problem

When running in type 4 mode, some versions of the DB2 Universal Driver trigger a generic exception for database failure rather than a specific error code that WebSphere Application Server can map to a stale connection exception. This problem occurs with versions of the driver that are associated with DB2 8.1 Fix Pack 6 or Fix Pack 7, and DB2 8.2.

Solution

Upgrade to DB2 Version 8.2 Fix Pack 1, equivalent to Version 8.1 Fix Pack 8, which provides a valid error code in the previously described scenario. WebSphere Application Server maps this error code to a StaleConnectionException, as expected.

Cannot access DB2 on Linux when using the DB2 Universal JDBC Driver

Symptom

In the WebSphere Application Server on Linux environment, applications that use the DB2 Universal JDBC Driver to access DB2 on Linux might not connect with the database. The database server can issue the following exceptions to the application server error log:

- `java.security.AccessControlException: Access denied (java.lang.RuntimePermission accessClassInPackage.sun.io)`
- *If you run 64-bit Linux:*

  `com.ibm.db2.jcc.b.SqlException: Failure in loading T2 native library db2jcct2`

Problem

The process for configuring DB2 on Linux to work with the Universal JDBC Driver is not complete.

Solution

- Verify that the setup requirements for the Java™ SDK 1.4.2 for the Linux platform are complete.
- Configure your development environment for building Java applications on Linux with DB2 JDBC support. See the application development topics for DB2 for more information.
- If you run DB2 on the Linux/IA64 platform and are using DB2 v8.1 Fix Pack 7A, perform the additional step that is described in the technote on DB2 UDB Version 8 FixPak 7a for Linux on IA64 reporting a

missing libdb.so.3 library. This step is necessary only for Fix Pack 7A. This step is not necessary for DB2 v8.1 Fix Pack 7 or earlier versions of DB2; this step is not necessary for DB2 v8.1 Fix Pack 8 or later versions of DB2.

Illegal conversion occurs on any VARCHAR FOR BIT DATA column in a container-managed persistent bean

When enterprise beans with container-managed persistent (CMP) types that have any VARCHAR FOR BIT DATA columns defined on a DB2 table are deployed in the DB2 universal JDBC type 4 driver to persist the data, an SQLException of illegal conversion is thrown at run time. This exception only occurs when you use the DB2 universal JDBC type 4 driver and with the deferPrepares property being set to true. When the deferPrepares property is set to true, the DB2 universal JDBC type 4 driver uses the standard JDBC data mapping.

Currently, the generated deployed code does not follow the standard JDBC specification mapping. The failure at execution time is because of a problem in the tool that prepared the enterprise beans for execution.

To avoid receiving this exception, choose one of the following options:
- Set the deferPrepares property to false in the data source configuration.
- Do not use the DB2 universal JDBC type 4 driver if your table has any VARCHAR FOR BIT DATA or LONG VARCHAR FOR BIT DATA columns. Refer to DB2 V8.1 readme for more details.

Data access problems - Microsoft SQL Server data source

This article provides troubleshooting tips for accessing Microsoft SQL Server data sources.

What kind of problem are you having accessing your Microsoft SQL Server database?

- "ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database"
- "Application fails with message stating ″Cannot find stored procedure...″ accessing a Microsoft SQL Server database"
- "ERROR CODE: SQL5042 when you run a Java application" on page 1565

ERROR CODE: 20001 and SQL STATE: HY000 accessing SQLServer database

The problem might be that the distributed transaction coordinator service is not started. Look for an error similar to the following example when attempting to access a Microsoft SQL Server database:

```
ERROR CODE: 20001
SQL STATE:  HY000
java.sql.SQLException: [Microsoft][SQLServer JDBC Driver]
[SQLServer]xa_open (0) returns -3
at com.microsoft.jdbc.base.BaseExceptions.createException(Unknown Source) ...
at com.microsoft.jdbcx.sqlserver.SQLServerDataSource.getXAConnection
(Unknown Source) ...
```

To confirm this problem:

1. Go to the Windows **Control Panel** and click **Services**(or click **Control Panel > Administrative Tools > Services**)
2. Verify whether the service **Distributed Transaction Coordinator** or **DTC** is started.
3. If not, start the Distributed Transaction Coordinator service.

Application fails with message stating ″Cannot find stored procedure...″ accessing a Microsoft SQL Server database

This error can occur because the stored procedures for the Java Transaction API (JTA) feature are not installed on the Microsoft SQL Server.

To resolve the problem, repeat the installation for the stored procedures for the JTA feature, according to the DataDirect Connect for JDBC driver installation guide.

ERROR CODE: SQL5042 when you run a Java application

This error can occur when you configure your application to run in the following manner:

1. You use a type 2 (application) driver running on the gateway to the OS 390
2. Your application is an XA application.

OS 390 does not use XA, but uses SPM. To resolve the problem:

1. Check your dbm cfg to see that the SPM is not started on the gateway.
2. Assign a port and set the *db2comm* variable to **TCPIP**.
3. Update the dbm cfg value *SPM_NAME* to use your machine name.
4. Start the SPM on the gateway.

Data access problems - Apache Derby database

This article provides troubleshooting tips for accessing Apache Derby databases.

What kind of problem are you having accessing your Apache Derby database?

- "Unexpected IOException wrapped in SQLException, accessing Apache Derby database"
- "The ″select for update″ operation causes table lock and deadlock when accessing Apache Derby"
- "ERROR XSDB6: Another instance of Apache Derby may have already booted the database ″database″""
- "Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Apache Derby databases″" on page 1566
- "Running an application causes a runtime exception which produces an unreadable message." on page 1566

Tip: Apache Derby errorCodes (2000, 3000, 4000) indicate levels of severity, not specific error conditions. In diagnosing Apache Derby problems, pay attention to the given sqlState value.

Unexpected IOException wrapped in SQLException, accessing Apache Derby database

This problem can occur because Apache Derby databases use a large number of files. Some operating systems, such as the Solaris Operating Environment, limit the number of files an application can open at one time. If the default is a low number, such as 64, you can get this exception.

If your operating system lets you configure the number of file descriptors, you can correct the problem by setting the number to a high value, such as 1024.

The ″select for update″ operation causes table lock and deadlock when accessing Apache Derby

If a select for update operation on one row locks the entire table, which creates a deadlock condition, there might be undefined indexes on that table. The lack of an index on the columns you use in the `where` clause can cause Apache Derby to create a table lock rather than a row level lock.

To resolve this problem, create an index on the affected table.

ERROR XSDB6: Another instance of Apache Derby may have already booted the database ″database″

This problem occurs because Apache Derby embedded framework only allows one Java virtual machine (JVM) to access the database instance at a time.

To resolve this problem:
1. Verify that you do not have other JDBC client programs, such as **ij** or **cview** running on that database instance, when WebSphere Application Server is running.
2. Verify that you do not use the same instance of the database for more than one data source or use the networkServer framework, which doesn't have this limitation.
3. If there are no connections to Apache Derby, delete the db.lck lock file. This file can be found in the directory where the Apache Derby database is mounted, under the schema directory. For example, if the database is mounted at /myDerbyDB, issue the command: rm /myDerbyDB/schemaName/db.lck

Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Apache Derby databases″

Error ″The version of the IBM Universal JDBC driver in use is not licensed for connectivity to Apache Derby databases″

At the client runtime, an error similar to the following occurs:

```
The version of the IBM Universal JDBC driver in use is not
licensed for connectivity to Apache Derby databases.  To connect
to this DB2 server, please obtain a licensed copy of the IBM DB2
Universal Driver for JDBC and SQLJ.  An appropriate license file
db2jcc_license_*.jar for this target platform must be installed to
the application classpath.  Connectivity to Apache Derby databases is
enabled by any of the following license files:
{ db2jcc_license_c.jar, b2jcc_license_cu.jar, db2jcc_license_cisuz.jar }
```

The problem occurs because an incorrect JDBC driver jar file name is specified in the class path for JDBC provider. For example, the jar file name may have an extra '_', as follows:

```
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license__cu.jar
```

To resolve the problem:
1. Correct the UNIVERSAL_JDBC_DRIVER_PATH jar file name in the JACL script
2. Restart the cluster.
3. Rerun the client.

Running an application causes a runtime exception which produces an unreadable message.

At client runtime, you may receive a message similar to the following: `Caused by: com.ibm.db2.jcc.a.SqlException: DB2 SQL error: SQLCODE: -1, SQLSTATE: 42X05, SQLERRMC: ANNUITYHOLDER20^T42X05`

The problem occurs because the property *retrieveMessagesfromServerOnGetMessage*, which is required by WebSphere Application Server, has not been set.

To resolve the problem, on the admininstrative console
1. Click **Resources -> JDBC Providers**
2. Click on a Apache Derby provider
3. Scroll down and click on **Data Sources**
4. Select your data source (or add a new one)
5. Scroll down and select **Custom Properties**
6. If the property *retrieveMessagesFromServerOnGetMessage* already exists, set its value to true. If the property does not exist, select **New**and add the property *retrieveMessagesFromServerOnGetMessage* with a value **true**
7. Rerun the client

The SystemOut.log will now generate readable messages so that you can resolve the underlying problem.

Data access problems - Sybase data source

This article provides troubleshooting tips for accessing Sybase data sources.

What kind of problem are you having accessing your Sybase database?

- ″Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode″ error.
- ″JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.″
- A container managed persistence (CMP) enterprise bean is causing exceptions.
- “Sybase JDBC data source fails with ″Incorrect URL format″ exception in IPv6 environment” on page 1568
- “Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors” on page 1568
- “Sybase requirements for using the escapes and DatabaseMetaData methods” on page 1569
- “Database deadlocks and XA_PROTO errors occur when using Sybase” on page 1569
- “Executing a stored procedure containing a SELECT INTO command causes exception” on page 1570
- “Error is incorrectly reported about IMAGE to VARBINARY conversion” on page 1570
- “JDBC 1.0 standard methods are not implemented and generate a SQL exception when used” on page 1570
- “Sybase transaction manager fails after trying to alleviate a deadlock error” on page 1570
- “Starting an XA transaction when the autoCommit value of the connection is false causes error” on page 1570
- “Sybase does not throw an exception when an incorrect database name is specified” on page 1571

″Sybase Error 7713: Stored Procedure can only be executed in unchained transaction mode″ error

This error occurs when either:

- The JDBC attempts to put the connection in **autocommit(true)** mode.
- A stored procedure is not created in a compatible mode.

To fix the **autocommit(true)** mode problem, let the application change the connection to chained mode using the **Connection.setAutoCommit(false)** mode, or use a **set chained on** language command.

To resolve the stored procedure problem, use the `sp_procxmode` *`procedure_name`* `"anymode"` command.

″JZ0XS: The server does not support XA-style transactions. Please verify that the transaction feature is enabled and licensed on this server.″

This error occurs when XA-style transactions are attempted on a server that does not have Distributed Transaction Management (DTM) installed.

To resolve this problem, use the instructions in the Sybase Manual titled: *Using Adaptive Server Distributed Transaction Management Features* to enable Distributed Transaction Management (DTM). The main steps in this procedure are:

1. Install the DTM option.
2. Check the `license.dat` file to verify that the DTM option is installed.
3. Restart the license manager.
4. Enable DTM in ISQL.
5. Restart the ASE service.

A container managed persistence (CMP) enterprise bean is causing exceptions

This error is caused by improper use of reserved words. Reserved words cannot be used as column names.

To correct this problem: Rename the variable to remove the reserved word. You can find a list of reserved words in the *Sybase Adaptive Server Enterprise Reference Manual; Volume 1: Building Blocks*, Chapter 4. This manual is available online at: http://manuals.sybase.com/onlinebooks/group-as/asg1250e/refman.

Sybase JDBC data source fails with ″Incorrect URL format″ exception in IPv6 environment

Problem

If you configure a Sybase JDBC data source through the administrative console and attempt to use it in an IPv6 environment, you can experience the following exception:

```
java.sql.SQLException: JZ0NE: Incorrect URL format
```

Cause

The Sybase JDBC drivers that are listed as selections for JDBC provider type in the administrative console do not support IPv6. The administrative console does not contain a pre-formatted template for the Sybase jConnect JDBC driver v6.0 EBF12884, which is the version required to connect to the database in an IPv6 environment.

However, you can still use the administrative console to define the Sybase jConnect JDBC driver v6.0 EBF12884.

Solution

Complete the following steps to define the Sybase jConnect JDBC driver v6.0 EBF12884:

1. In the administrative console, go to **Resources** > **JDBC Providers** and click **New**.
2. Select **User-defined** for the database type. This selection triggers the console to set your provider type to User-defined JDBC provider and implementation type to User-defined.
3. Click **Next** to go to the JDBC provider general configuration page.
4. In the Class path field, replace the default JAR file name with `jconn3.jar`.
5. For Implementation class name:
   - For a data source implementation that supports only one phase transactions, input
     `com.sybase.jdbc3.jdbc.SybConnectionPoolDataSource`
   - For an implementation that supports two phase transactions, input
     `com.sybase.jdbc3.jdbc.SybXADataSource`
6. Click **Apply**. Then click the Data sources link.
7. Click **New** to define the general properties of your data source.
8. Click **Apply** after defining the general data source properties. Then click the Custom properties link.
9. Define the following properties as custom properties. To set these properties, click **New**. Input the property name and a valid value for each one.
   - databaseName
   - serverName
   - portNumber

Executing the DatabaseMetaData.getBestRowIdentifier() method in an XA transaction causes errors

Executing the DatabaseMetaData.getBestRowIdentifier() method while in an XA transaction causes the following errors:

```
SQL Exception: The 'CREATE TABLE' command is not allowed within a
multi-statement transaction in the 'tempdb' database.  Calling
DatabaseMetaData.getBestRowIdentifier()
```

Currently, this method fails when using Sybase. This problem occurs with other methods as well, including:
- getBestRowIdentifier();
- getVersionColumns();
- getTablePrivileges();
- getProcedureColumns();
- getPrimaryKeys();
- getIndexInfo();
- getImportedKeys();
- getExportedKeys();
- getCrossReference();
- getColumns();
- getColumnPrivileges();

Case 10880427 has been opened with Sybase to resolve this problem.

Sybase requirements for using the escapes and DatabaseMetaData methods

To use the escapes and DatabaseMetaData methods, you must install stored procedures on the Adaptive Server Enterprise or Adaptive Server Anywhere database where you want to use these methods. These stored procedures are also required by some of the connection methods.

To check for the presence of LOCATE ():
1. Open a Sybase **isql** command prompt.
2. Type the command **use master**.
3. Type the command **go**.
4. Type the SQL command and select * from jdbc_function_escapes.
5. Type the command **go**.

The following appears:

```
escape_name      map_string
 --------------------------------
 abs                    abs(%1)
 acos                  acos(%1)
 asin                   asin(%1)
 atan                   atan(%1)
 atan2                 atn2(%1, %2)
 ceiling                ceiling(%1)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::

locate charindex ((convert (varchar, %1)), (convert (varchar, %2)))
```

If the function does not exist, upgrade jConnect to at least Version 5.2 EBF 10635 and run the following command:

```
java IsqlApp -U sa -P -S jdbc:sybase:Tds:hostname:4100 -I %JDBC_HOME%\sp\sql_server12.sql -c go
```

Database deadlocks and XA_PROTO errors occur when using Sybase

When using Sybase with the IBM WebSphere Application Server, do one of the following to prevent database deadlocks and errors:
- Change the transaction isolation level on the connection to TRANSACTION_READ_COMMITTED. Set the isolation level on the connection for unshareable connections or, for shareable connections, define the isolation levels in the resource reference for your data source using an assembly tool.
- Modify Sybase by doing one of the following:
  - If you want to use the existing tables, modify the table locking scheme using the **alter table *table name* lock datarows** command to get a row lock level granularity.
  - If you want to set the system-wide locking scheme to data rows, all subsequently created tables inherit that value and have a locking scheme of data rows.

Note: You must drop your original databases and tables.

Executing a stored procedure containing a SELECT INTO command causes exception

An attempt to execute a stored procedure containing a **SELECT INTO** command results in the following exception:

```
SVR-ERROR: SQL Exception SELECT INTO command not allowed within multi-statement transaction
```

Case 10868947 has been opened with Sybase to resolve this problem.

Error is incorrectly reported about IMAGE to VARBINARY conversion

The following error is incorrectly reported:

```
com.sybase.jdbc2.jdbc.SybSQLException: Implicit conversion from data type 'IMAGE' to 'VARBINARY' is not allowed.
 Use the CONVERT function to run this query.
```

The error is about a VARBINARY column only and causes confusion if you also have an IMAGE column.

Do one of the following to work around this problem:
- Use a PreparedStatement.setBytes() method instead of a PreparedStatement.setBinaryStream() method
- Use a LONG VARBINARY for the column type if you want to continue using the setBinaryStream() method. You might want to make this change because the size limit for VARBINARY is 255 bytes.

For example:

```
// ***************CORRECTION****************************
         // setBinaryStream fails for column type of VARBINARY , use setBytes() instead
         //stmt4.setBinaryStream(8,new java.io.ByteArrayInputStream(tempbyteArray),tempbyteArray.length);
         stmt4.setBytes(8,tempbyteArray);
```

JDBC 1.0 standard methods are not implemented and generate a SQL exception when used

The following JDBC 1.0 standard methods are not implemented and generate a SQL exception when used:
- ResultSetMetaData.getSchemaName()
- ResultSetMetaData.getTableName() (implemented only for text and image datatypes)
- ResultSetMetaData.getCatalogName()

Sybase transaction manager fails after trying to alleviate a deadlock error

If an application encounters a deadlock, Sybase detects the deadlock and throws an exception. Because of this detection, the transaction manager calls an `xa_end` with a `TMFAIL` in it.

The call succeeds, but causes another Sybase exception, `XAERR_PROTO`. This exception only appears in the error log and does not cause any functional problems. All applications should continue to run, therefore no workaround is necessary.

Case 10869169 has been opened with Sybase to resolve this problem.

Starting an XA transaction when the autoCommit value of the connection is *false* causes error

The exception thrown is `javax.transaction.xa.XAException` with stack trace similar to the following:

```
at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:711)
 at com.sybase.jdbc2.jdbc.SybXAResource.sendRPC(SybXAResource.java:602)
 at com.sybase.jdbc2.jdbc.SybXAResource.start(SybXAResource.java:312)
```

This problem affects you when you do both local and global transactions. If, in a local transaction, the autoCommit default value is set to *false*, and a global or XA transaction starts (either a user transaction started by you, or a container transaction started by a container), the exception occurs.

This problem is a Sybase bug as the start() method can fail unexpectedly, regardless of the value of autoCommit. Currently, there is no workaround for this problem, therefore it is not recommended that you mix local and global transactions. Case 10880792 has been opened to resolve this problem.

Sybase does not throw an exception when an incorrect database name is specified

Verify that your database name is correctly entered on the data source properties.

Most databases (DB2, Oracle, Informix , MS SQL Server and Cloudscape) throw an exception when the database specified does not exist. But Sybase does not throw an exception when an incorrect database name is specified. Sybase generates an SQL warning and then connects to the default database. If you misspell the requested database name, Sybase connects you to the master or the default database where the table you requested is not found.

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes). If you do not find your problem listed there, contact IBM Support.

JDBC trace configuration

If your application displays JDBC-related exception messages, activate the JDBC trace service. The resulting log text can help you identify the problem.

Trace strings for JDBC data sources

Turn on tracing for most database JDBC implementations through the administrative console; see the article Enabling trace at server startup for instructions.

This method activates JDBC trace for all applications that run in the server you specify. Identify your database type by selecting the trace group WAS.database and typing one of the following trace strings in the console:

- **com.ibm.ws.database.logwriter** Trace string for databases that use the GenericDataStoreHelper. You can also use this trace string for unsupported databases.
- **com.ibm.ws.db2.logwriter** Trace string for DB2 databases.
- **com.ibm.ws.oracle.logwriter** Trace string for Oracle databases.
- **com.ibm.ws.derby.logwriter** Trace string for Derby databases.
- **com.ibm.ws.informix.logwriter** Trace string for Informix databases.
- **com.ibm.ws.sqlserver.logwriter** Trace string for Microsoft SQL Server databases.
- **com.ibm.ws.sybase.logwriter** Trace string for Sybase databases.

A few JDBC drivers require that you set trace differently, at the data source level. These drivers include:

- Microsoft SQL Server JDBC driver
- DataDirect Connect for JDBC driver for MS SQL Server

Configuring trace for these drivers through the WAS.database group results in corrupt trace information. The application server sets trace for the group at the server level, causing the trace service to begin only after your application establishes an initial connection. Because that first connection does not carry trace information, re-use of it is never tracked. Consequently the application cannot accurately match trace information to connection use.

Set trace for the previously mentioned JDBC drivers through data source custom properties. For example, use the spyAttributes custom property to enable JDBC trace for the DataDirect Connect for JDBC driver.

Consult your driver documentation for details on the custom property that enables trace for your JDBC implementation.

Additional resources

If the JDBC tracing service cannot help you isolate and fix your problem, consult the IBM Support web site for WebSphere Application Server. Use the site search function to find current information on known problems and their resolutions. Locating the right troubleshooting tip can save time that you might otherwise spend on opening and tracking a PMR.

Configuring the connection validation timeout

You can configure a timeout for connection validation by the Java Database Connectivity (JDBC) driver through a data source custom property in the data source configuration panels.

About this task

You can choose between validating connections with the JDBC driver or by having the application server run a SQL query. Select one or both of the following connection pretest attributes:

- Validate new connections
- Validate existing pooled connections

By default, connection validation is disabled. When you save the configuration for the data source, the administrative console supplies only the option that is selected. The administrative console will select validation by timeout or validation by a query, but if validation is not enabled then the application server will select neither option.

1. Open the administrative console.
2. Go to the **WebSphere Application Server Data Source properties** panel for the data source.
   a. Select **Resources** → **JDBC** → **Data Sources** → ***data_source***
   b. Select **WebSphere Application Server Data Source properties**.
3. Go to the **Connection Validation Properties** section.
4. Select the type of connections that the application server will validate.
   - Select **Validate new connections**. This option specifies that the connection manager tests newly created connections to the database.
   - Select **Validate existing pooled connections**. This options specifies that the connection manager tests the validity of pooled connections before returning them to applications.
   - You can also select both options

     Note: You must make a selection here. If you do not select one or both of these options, you will not be able to select **Validation by JDBC Driver**. The **Validation by JDBC Driver** timeout feature is only available for JDBC providers that comply with the JDBC 4.0 specification.

     .
5. Click **Validation by JDBC Driver**. The application server issues a warning if **Validation by JDBC driver** is configured and the JDBC driver does not implement JDBC 4.0, or if the Connection.isValid method raises an error.

   Note: Connection validation by SQL query is deprecated. Use validation by JDBC Driver instead.
6. Enter the timeout value in the input box. The timeout value is in seconds.

   Note: If retries are configured, meaning the retry interval is not set to 0, for **Validate new connections** or **Validate existing pooled connections**, then the full value of the timeout applies to each retry. For each retry, the application server waits for the retry interval. Then the JDBC driver uses the full value of the timeout to validate the connection

7. Save the data source configuration.

What to do next

If you are modifying an existing data source, restart your server for this change to go into effect. If this is a new data source, restarting the server is not necessary.

Deploying data access applications

Frequently, deploying data access applications involves more than installing your WAR or EAR file onto a server. Deployment can include tasks for configuring your application to use the data access resources of the server and overall run-time environment.

Before you begin

You can only deploy application code that is assembled into the appropriate modules. The topic Assembling data access applications provides guidelines for this process.

About this task

Perform the following steps if your application requires access to a relational database (RDB). If your application requires access to a different type of enterprise information system (EIS), such as an object-oriented database or the Customer Information Control System (CICS), consult the topics “Resource adapters” on page 1315 and Accessing data using Java EE Connector Architecture connectors.

1. If your RDB configuration does not already exist:
   a. Create a database to hold the data.
   b. Create tables required by your application.
      If your application uses CMP entity beans to access the data
      You can create the tables using the data definition language (DDL) generated from the enterprise bean configuration. For more information, see Recreating database tables from the exported table data definition language.
      If your application uses BMP entity beans, or *does not* use entity beans
      You must use your database server interfaces to create the tables.

      You can also use the EJB to RDB Mapping wizard of an assembly tool to create your database tables for either type of entity bean. Select the top-down mapping option in the wizard. Keep in mind, however, that this option does not give you direct control in naming the RDB elements or choosing column types. Additionally, because the top-down process is automatic, it might not provide mappings to reflect the precise relationships that you intend.

      If you use Rational Application Developer, consult the information center about the mapping wizard. To learn about all of your assembly tool options, consult the Assembly tools article in this information center.
   c. Check “Data source minimum required settings, by vendor” on page 1433 to see any database vendor requirements for connecting to an application server.
2. If necessary, map your entity beans to the database tables through the meet-in-the-middle mapping option of an assembly tool. This step is necessary only if you did not create your database schema through the top-down mapping option, did not generate your mapping relationships through bottom-up mapping, or did not generate mappings during the application assembly process. For information on the top-down mapping option refer to the information center for Rational Application Developer.
3. Install your application onto the application server. Consult Installing enterprise application files. When you install the application, you can alter data access settings that were made during application assembly, or set them for the first time if they were omitted from the assembly process. These settings include resource bindings and resource authentication aliases, which are addressed in the following substeps:

a. Bind application resource references to the data sources, or other resource objects, that provide database connectivity. For details on the concept of binding, see the Data source lookups for enterprise beans and Web modules topic.

   Note: After deployment, you can use the WebSphere Application Server administrative console to alter resource bindings. Click **Applications** → **Application Types** → **Webphere enterprise applications** → ***application_name***, and select the link to the appropriate mapping page. For example, if you want to alter the binding of an EJB module resource, you might click **Map data sources for all 2.x CMP beans** . For a Web module resource, click **Resource references**.

b. Define authentication alias data for resources that must be authenticated with the backend through *container-managed* authorization. In this security configuration, WebSphere Application Server performs EIS signon for data source or connection factory connections. Consult the "J2EE connector security" on page 1426 topic for detailed reference on resource authentication.

4. Start the deployed application files using the administrative console , the wsadmin startApplication command, or your own Java program.
5. Save the changes to your administrative configuration.
6. Test the application. For example, point a Web browser at the URL for a deployed application and examine the performance of the application.

Results

When you deploy an application that uses a DB2 UDB for i5/OS back-end database, you might find the following exception in the SystemOut.log file:

```
PMGR6022E: Error using adapter to create or execute an Interaction
```

This type of error indicates that you deployed an application with Container-managed persistence (CMP) enterprise beans that were originally configured to access a DB2 database on Windows, Linux, or a supported UNIX system. Using the administrative console, uninstall the affected CMP applications, then reinstall the applications with the new database setting. Remember to select **Deploy enterprise beans**; on the EJB deploy panel, select the appropriate version of your DB2 UDB for i5/OS database.

What to do next

If the application does not perform as desired, update the application, then save and test it again.

Available resources

Use this page to select configured resources that you want to bind to the resource references of the enterprise beans or web modules in your application.

To view this administrative console page:

1. Click **Applications** → **Application Types** → **Websphere enterprise applications** → ***application_name***.
2. Click the link for any of these resource configuration pages:
   - **Resource references**
   - **Map data sources for all 2.x CMP beans**
   - **Provide default data source mapping for modules containing 2.x entity beans**
3. Locate the table row of the EJB or web module that you want to map to a different resource.
4. Within the row, locate the JNDI name of the resource that is currently bound to the EJB or web module.
5. Click **Browse**.

   You now see **Available resources**.

Each table row corresponds to a resource that you can bind to your enterprise bean or web module.

Select
Select the resource that you want to bind to the resource reference of your module.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the resource that you want to bind to the resource reference of your module.

Data type String

Scope
The scope of the resource. Note that this administrative console page displays only resources that are configured for a scope at which your application operates.

Description
The text description of the resource.

Map data sources for all 1.x CMP beans
Use this page to designate how the container-managed persistence (CMP) 1.x beans of an application map to data sources that are available to the application.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Map data sources for all 1.x CMP beans**.

Guidelines for using this administrative console page:

- The table depicts the 1.x CMP bean contents of your application.
- Each table row corresponds to a CMP bean within a specific EJB module. A row shows the JNDI name of the data source mapping target of the bean *only* if you bound them together during application assembly or installation. For every data source that is displayed, you see the corresponding security configuration.
- To set your mappings:
  1. Select a row. Be aware that if you check multiple rows on this page, the data source mapping target that you select in step 2 applies to all of those CMP beans.
  2. Click **Browse** to select a data source from the new page that is displayed, the Available Resources page. The Available Resources page shows all data sources that are available mapping targets for your CMP beans.
  3. Click **Apply**. The console displays the 1.x CMP bean data sources page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
  4. *Before* you click **OK** to save your new configuration, set the security parameters for the data source. Use the following steps.
- To specify data source security settings:
  1. Select one or more rows in the table.
  2. Type in a user name and password that comprise the authentication alias for signing on to the data source. If these entries are not listed in the application Java Platform, Enterprise Edition (Java EE) Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the information center topic on managing Java EE Connector Architecture authentication data entries for more information.
  3. Click **Apply** that immediately follows the user name and password input fields.
- Repeat all of the previous steps as necessary.
- Click **OK** to save your settings.

Select
Select the check boxes of the rows that you want to edit.

EJB
The name of an enterprise bean in the application.

EJB Module
The name of the module that contains the enterprise bean.

URI
Specifies location of the module relative to the root of the application EAR file.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the data source that is configured for the enterprise bean.

Data type String

User name
The user name and password that comprise the authentication alias for securing the data source.

Map default data sources for modules containing 1.x entity beans
Use this page to set the default data source mapping for EJB modules that contain 1.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 1.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Map default data sources for modules containing 1.x entity beans**.

Guidelines for using this administrative console page:

- The page displays a table that depicts the EJB modules in your application that contain 1.x CMP beans.
- Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target of the EJB module *only* if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.
- To set your default data source mappings:
  1. Select a row. Be aware that if you check multiple rows on this page, the data source mapping target that you select in step 2 applies to all of those EJB modules.
  2. Click **Browse** to select a data source from the new page that is displayed, the Available Resources page. The Available Resources page shows all data sources that are available mapping targets for your EJB modules.
  3. Click **Apply**. The console displays the 1.x entity bean data sources page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
  4. *Before* you click **OK** to save your new configuration, set the security parameters for the data source. Use the following steps.
- To specify security settings for the default data source:
  1. Select a row. Be aware that if you check multiple rows on this page, the security settings that you select later apply to all of those data sources.
  2. Type in a user name and password that comprise the authentication alias for signing on to the data source. If these entries are not listed in the application Java Platform, Enterprise Edition (Java EE) Connector (J2C) authentication data list, you must input them into the list after saving your settings on this page. Read the information center topic on managing Java EE Connector Architecture authentication data entries for more information.

3. Click **Apply** that immediately follows the user name and password input fields.

- Repeat all of the previous steps as necessary.
- Click **OK** to save your work.

Select
Select the check boxes of the rows that you want to edit.

EJB Module
The name of the module that contains the 1.x enterprise beans.

URI
Specifies location of the module relative to the root of the application EAR file.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Data type String

User name
The user name and password that comprise the authentication alias for securing the data source.

Map data sources for all 2.x CMP beans settings
Use this page to map container-managed persistence (CMP) 2.x beans of an application to data sources that are available to the application.

To view this administrative console page, click **Applications** → **Application Types** → **Websphere enterprise applications** → ***application_name*** → **Map data sources for all 2.x CMP beans**.

Each table row corresponds to a CMP bean within a specific EJB module. A row shows the JNDI name of the data source mapping target of the bean only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI names
Specify the Java Naming and Directory Interface (JNDI) name for multiple EJB modules. Select one or more EJB modules from the table, and select a JNDI name from this list to configure the EJB modules with that JNDI name.

Data type Drop-down list

Set Authorization Type
Specify the authorization type for securing the data source. Select one or more EJB modules from the table to set the authorization type.

Select either **Container** or **Application** from the displayed list. Container-managed authorization indicates that WebSphere Application Server performs signon to the data source. Application-managed authorization indicates that the enterprise bean code performs signon.

Modify Resource Authentication Method
Specify the authorization type and the authentication method for securing the data source. Select one or more EJB modules from the table to modify the resource authentication method.

You can choose between the following authentication methods:

- **None**:
  1. Determine which data source configurations to designate with no authentication method.

2. Select the appropriate table rows.
3. Select **None** from the list of authentication method options that precede the table.
4. Click **Apply**.

- **Use default method (many-to-one mapping)**:
  1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
  2. Select the appropriate table rows.
  3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
  4. Select an authentication data entry or alias from the list.
  5. Click **Apply**.
- **Use Kerberos authentication**: Specifies to use the Kerberos authentication method.
  1. Ensure that you have configured the Kerberos authentication mechanism in the application server.
  2. Select the appropriate table row.
  3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Use trusted connections (one-to-one mapping)**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Ensure that the database to which the modules will connect is configured for trusted connections.
  4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.
  5. Select an application login configuration from the list.
  6. Click **Apply**.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Custom login configuration**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select

Select the check boxes of the rows that you want to edit.

EJB
The name of an enterprise bean in the application.

EJB Module
The name of the module that contains the enterprise bean.

URI
Specifies location of the module relative to the root of the application EAR file.

Target resource JNDI name
Specifies the resource to which the CMP bean is bound.

Resource authorization
Specifies the current setting for the resource authorization type.

Modify this setting with **Set authorization type**.

Map data sources for all 2.x CMP beans
Use this page to set the default data source mapping for EJB modules that contain 2.x container-managed persistence (CMP) beans. Unless you configure individual data sources for your 2.x CMP beans, this default mapping applies to all beans within the module.

To view this administrative console panel, click **Applications** → **Application Types** → **Websphere enterprise applications** → ***application_name*** → **Map data sources for all 2.x CMP beans** .

This panel displays a table that depicts the EJB modules in your application that contain 2.x CMP beans. Each table row corresponds to a module. A row shows the JNDI name of the data source mapping target of the EJB module only if you bound them together during application assembly. For every data source that is displayed, you see the corresponding security configuration.

Set Multiple JNDI Names
Specifies the JNDI name to bind to one or more modules. Select one or more modules, click **Set Multiple JNDI Names**, and select the JNDI name for the resource to which you would like to bind the module.

Set Authorization Type
Specifies the authorization type that you to use for the modules. Select one or more modules, click **Set Authorization Type**, and select the authorization type.

You can choose:

- Per application - indicates that the enterprise bean code performs signon.
- Container - indicates that the application server performs signon to the data source.

Modify Resource Authentication Method
Specifies the resource authentication method for the modules that you have configured with container-managed authorization. Select one or more modules, click **Modify Resource Authentication Method**, and select the authentication method.

You can choose between the following authentication methods:

- **None**:
  1. Determine which data source configurations to designate with no authentication method.
  2. Select the appropriate table rows.
  3. Select **None** from the list of authentication method options that precede the table.
  4. Click **Apply**.
- **Use default method (many-to-one mapping)**:

1. Determine which data source configurations to designate with the WebSphere Application Server DefaultPrincipalMapping login configuration. Apply this option to each data source individually if you want to designate different authentication data aliases. See the information center topic on J2EE Connector security for more information on the default mapping configuration.
2. Select the appropriate table rows.
3. Select **Use default method (many-to-one mapping)** from the list of authentication method options that precede the table.
4. Select an authentication data entry or alias from the list.
5. Click **Apply**.

- **Use Kerberos authentication**: Specifies to use the Kerberos authentication method.
  1. Ensure that you have configured the Kerberos authentication mechanism in the application server.
  2. Select the appropriate table row.
  3. Select **Use Kerberos authentication** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Use trusted connections (one-to-one mapping)**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Ensure that the database to which the modules will connect is configured for trusted connections.
  4. Select **Use trusted connections (one-to-one mapping)** from the list of authentication method options that precede the table.
  5. Select an application login configuration from the list.
  6. Click **Apply**.

  The application server will attempt to verify that you are connecting to the correct type of database when you select this option.
- **Custom login configuration**:
  1. Determine which data source configurations to designate with a custom Java Authentication and Authorization Service (JAAS) login configuration. See the information center topic on J2EE Connector security for more information on custom JAAS login configurations.
  2. Select the appropriate table row.
  3. Select **Use custom login configuration** from the list of authentication method options that precede the table.
  4. Select an application login configuration from the list.
  5. Click **Apply**.
  6. To edit the properties of the custom login configuration, click **Mapping Properties** in the table cell.

Select
Select the check boxes of the rows you want to edit.

EJB Module
Specifies the name of the module that contains the 2.x enterprise beans.

URI
Specifies location of the module relative to the root of the application EAR file.

JNDI name
Specifies the Java Naming and Directory Interface (JNDI) name of the default data source for the EJB module.

Data type String

Resource authorization
Specifies the authorization type and the authentication method for securing the data source.

Extended Datasource Properties
When selected, you will be directed to a panel on which you can specify extended properties that the module can use for the DB2 data source.

The application server will attempt to verify that you are connecting to the correct type of database when you select this option.

Chapter 8. Messaging resources

Managing messaging with the default messaging provider

This topic is the entry point into a set of topics about enabling WebSphere Application Server applications to use messaging resources provided by the default messaging provider.

Before you begin

For messaging between application servers, perhaps with some interaction with a WebSphere MQ system, you can use the default messaging provider as described in this topic. To integrate WebSphere Application Server messaging into a predominantly WebSphere MQ network, you can use the WebSphere MQ messaging provider. You can also use a third party messaging provider. To choose the provider that is best suited to your needs, see Choosing a messaging provider.

About this task

The default messaging provider is installed and runs as part of WebSphere Application Server, and is based on service integration technologies.

The default messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as “unified” or “common” interfaces). This enables applications to use common interfaces for both point-to-point and publish/subscribe messaging. This also enables both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, this approach is recommended for new applications. The domain-specific interfaces are supported for backwards compatibility for applications developed to use domain-specific queue interfaces, as described in section 1.5 of the JMS 1.1 specification.

You can use the WebSphere Application Server administrative console to configure JMS resources for applications, and can manage messages and subscriptions associated with JMS destinations.

WebSphere Application Server Version 5.1 Java EE applications can use messaging resources of the default messaging provider in WebSphere Application Server Version 7.0. This JMS interoperation from WebSphere Application Server Version 5.1 to later versions is enabled and managed by a WebSphere MQ client link created on the WebSphere Application Server Version 7.0 node. This JMS interoperation is only intended as an aid to the migration from the embedded messaging in WebSphere Application Server Version 5.1 to the default messaging provider in WebSphere Application Server Version 7.0.

Note: Java EE applications running under WebSphere Application Server Version 7.0 can use messaging resources of Version 5 embedded messaging without any need for a WebSphere MQ client link.

For more information about using the default messaging provider of WebSphere Application Server, see the following topics:

- Learning about the default messaging provider
- “Configuring resources for the default messaging provider” on page 1584
- “Interoperating with a WebSphere MQ network” on page 1612
- Migrating from WebSphere Application Server Version 5 embedded messaging
- “Managing WebSphere Application Server Version 5.1 JMS use of messaging resources in later versions of the product” on page 1646
- “Configuring the messaging engine selection process for JMS applications” on page 1656
- “Managing messages and subscriptions for JMS destinations” on page 1657
- “Using JMS from standalone clients to interoperate with service integration resources” on page 1659
- “Using JMS from a third party application server” on page 1666

Configuring resources for the default messaging provider

Use the following tasks to configure JMS connection factories, activation specifications, and destinations for the default messaging provider.

About this task

Use these tasks to configure administrative JMS resources provided by the default messaging provider.

These administrative JMS resources are in addition to any temporary JMS destinations created by applications.

- "Listing JMS resources for the default messaging provider"
- Configuring a JMS connection factory.
- Configuring a JMS queue connection factory.
- Configuring a JMS topic connection factory.
- Configuring a JMS queue destination.
- Configuring a JMS topic destination.
- "Configuring a JMS activation specification for the default messaging provider" on page 1591

Configuring resources for the default messaging provider by using the wsadmin tool

You can use these commands to manage JMS resources for the default messaging provider.

About this task

These commands provide an alternative to using the administrative console or using the more complex syntax of wsadmin and JACL.

1. Open a wsadmin command session in local mode For example:

   ```
   wsadmin –conntype none
   ```

   Note: The wsadmin scripting client is run from Qshell. For more information, see Configuring Qshell to run WebSphere Application Server scripts.
2. Type the command as indicated in the related reference topics.

Listing JMS resources for the default messaging provider

Use the WebSphere Application Server administrative console to list JMS resources for the default messaging provider, for administrative purposes.

About this task

You use the WebSphere Application Server administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:

- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue
- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:

- **Provider-centric navigation**. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS providers** is provider-centric.

- **Resource-centric navigation**. This lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → ***resource_type***, where *resource_type* is one of the resource types listed above is resource-centric.

You can use either of these navigation pathways to locate resources of any type.

- Optional: To use provider-centric navigation, for example to navigate to a specified connection factory, use the administrative console to complete the following steps:
  1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
  2. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
  3. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.
  4. From the connection factories collection, select the required connection factory.
- Optional: To use resource-centric navigation, for example to navigate to a specified connection factory, use the administrative console to complete the following steps:
  1. In the navigation pane, click **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.
  2. From the connection factories collection, select the required connection factory.

Results

You can now view and work with the resource's properties.

Configuring JMS resources for point-to-point messaging

Configure connection factories, queues and service integration bus destinations for point-to-point messaging.

About this task

For an application to use point-to-point messaging with JMS queues, you need to configure the following JMS resources. These JMS resources depend on the corresponding configuration of service integration resources, including a service integration bus and a queue. For more information about defining these resources, see the related tasks.

- Configure a connection factory.

  Use the connection factory type that matches the JMS level and domain pattern in which an application is developed. For example, use a domain-independent JMS connection factory for a JMS application developed to use JMS 1.1 domain-independent interfaces, and use a JMS queue connection factory for a JMS application developed to use domain-specific queue interfaces.
- Configure a queue.

  A *JMS queue* is an administrative object that encapsulates the name of a queue destination on a service integration bus. Applications find the JMS queue by looking up its name in the JNDI namespace.
- Configure a queue destination.

  For each JMS queue, define a bus destination of type queue on the appropriate service integration bus.

Configuring JMS resources for publish/subscribe messaging

Use this task to configure a JMS resources for publish/subscribe messaging.

About this task

For an application to use publish/subscribe messaging with JMS topics, you need to configure the following JMS resources. These JMS resources depend on the corresponding configuration of service integration resources, including a service integration bus and topicspace. For more information about defining these resources, see the related tasks.

- A JMS connection factory

  You should use the connection factory type that matches the JMS level and domain pattern in which an application is developed. For example, use a domain-independent JMS connection factory for a JMS application developed to use JMS 1.1 domain-independent interfaces, and use a JMS topic connection factory for a JMS application developed to use domain-specific topic interfaces.

  If durable subscriptions are to be used by the application, then you need to set the durable subscription properties on the connection factory.
- A JMS topic

  A *JMS topic* is an administrative object that encapsulates the name of a topic and a topic space on a service integration bus. Applications can obtain the JMS topic by looking its name up in the JNDI namespace.
- Service integration resources For each JMS topic, you need to define a bus destination as a topic space on a service integration bus, and assign the JMS topic to a topic name within that topic space.

Configuring a unified JMS connection factory for the default messaging provider

Use this task to configure a unified JMS connection factory for applications that use the JMS 1.1 domain-independent (unified) interfaces.

About this task

A unified[9] JMS connection factory can be used for both point-to-point and publish/subscribe JMS messaging. With JMS 1.1, this approach is preferred to the domain-specific queue connection factory and topic connection factory.

To configure a JMS connection factory for the default messaging provider, use the administrative console to complete the following steps. This task contains an optional step for you to create a new connection factory if you have not already created the connection factory that you want to configure.

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure a unified connection factory.
3. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Connection factories**. This displays any existing connection factories in the content pane.
5. If the connection factory is for use by client applications, display the properties of the JMS connection factory. If you want to display an existing JMS connection factory, click one of the names listed.

   Alternatively, if you want to create a new JMS connection factory, click **New**, then specify the following required properties:

   Name Type the name by which the connection factory is known for administrative purposes.

   JNDI name
   Type the JNDI name that is used to bind the connection factory into the name space.

9. The term “unified” refers to the support of both queues and topics by the same connection factory. This is similar to the JMS 1.1 domain-independent interfaces (referred to as the “common interfaces” in the JMS specification).

Bus name

Select the name of the service integration bus to which the connection factory is to create connections. This service integration bus hosts the destinations that the JMS queues and topics represent.

6. Review the other properties for the JMS connection factory, to check that the defaults are suitable. If needed, change the values according to your needs.

   If the connection factory is for use by client applications running outside of an application server, you need to specify suitable provider endpoints. For more information about configuring provider endpoints, see “Configuring a connection to a non-default bootstrap server” on page 1598.

   By default connections created using the connection factory in the server containers (for example, from an enterprise bean) are pooled using Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection pooling. You can modify the connection pool settings for the connection factory by selecting the Connection pool properties link in the Additional Properties section of the panel. For more information about changing the connection pool properties, see Changing connection pool settings with the wsadmin tool.
7. Click **OK**.
8. Save your changes to the master configuration.

Administrative properties for JMS connections to a bus:

This topic describes properties that you can configure to enable workload management of connections to a service integration bus for JMS applications.

The same properties can also be used to control the client connection topology. For example, connection options can be specified such that client applications only connect to a set of client serving messaging engines and never to the set of destination serving messaging engines in a bus.

The properties for connecting JMS applications to a bus are for administrator interest; the JMS applications are unconcerned with connections to the bus, beyond using a JMS connection factory or JMS activation specification (for message-driven beans).

The general aim of connecting to a bus is to connect to a suitable messaging engine that provides the message point for a JMS destination that the application wants to use. Applications running inside an application server can locate a suitable messaging engine and connect directly to the selected messaging engine. Client applications running outside of an application server cannot locate a suitable messaging engine themselves, these clients must use a bootstrap server to locate a suitable messaging engine on behalf of the client application.

When an application connects to the bus, the bus chooses a suitable messaging engine based on administrative properties of the JMS connection factory or activation specification that the application uses. For maximum connection flexibility, you can leave most properties to default, the only required connection property is the name of the bus that the application is to connect to.

The bus uses the following general process to choose a suitable messaging engine, based on the value you select for the Connection proximity property. If you understand this process, you can better configure the properties that control how the bus chooses messaging engines.

- If a **Target group** is specified then the process checks the nearest messaging engine that supports the required **Remote transport chain** and is a member of the target group in the bus. If the messaging engine is within the specified **Connection proximity** it is chosen as a suitable messaging engine for the application to connect to.
- If a **Target group** is not specified then the process checks the nearest messaging engine that supports the required **Remote transport chain** in the bus. A messaging engine in the same server is nearer than

a messaging engine in the same host, which is nearer than a messaging engine in another host. If the messaging engine is within the specified **Connection proximity** it is chosen as a suitable messaging engine for the application to connect to.

- If the selected messaging engine is not within the specified **Connection proximity**, then the **Target significance** is used. If the **Target significance** is set to Required, then no connection is possible and the connection request is rejected with no suitable messaging engine being available. If the **Target significance** is set to Preferred then the target group is ignored and the nearest messaging engine that supports the required **Remote transport chain** is used. If no messaging engine is found then the connection request is rejected with no suitable messaging engine being available.

The following rules are used to test the connection proximity for a selected messaging engine:

- If the Connection proximity value is Bus, then the selected messaging engine is used.
- If the Connection proximity value is Host, and the selected messaging engine is in the same host as the application (or bootstrap server), then the selected messaging engine is used. Otherwise, one of the following options is chosen.
  - If the selected messaging engine is not in the same host as the application (or the bootstrap server), and the **Target significance** is set to Required, then no connection is possible and the connection request is rejected with no suitable messaging engine being available.
  - If the **Target significance** is set to Preferred then the nearest messaging engine - in the same host - that supports the required **Remote transport chain** is used
  - If no suitable messaging engine is found, then the connection request is rejected.
- If the Connection proximity value is Server, and the selected messaging engine is in the same server as the application (or bootstrap server), then the selected messaging engine is used. Otherwise, one of the following options is chosen.
  - If the selected messaging engine is not in the same server as the application (or is in the bootstrap server), and the **Target significance** is set to Required, then no connection is possible and the connection request is rejected with no suitable messaging engine being available.
  - If the **Target significance** is set to Preferred then the nearest messaging engine - in the same server - that supports the required **Remote transport chain** is used
  - If no suitable messaging engine is found, then the connection request is rejected.

When a connection is made to a messaging engine in the same server as the application, the connection is made directly through memory, so the **Remote transport chain** is ignored.

Configuring a JMS queue connection factory for the default messaging provider

Use this task to configure a JMS queue connection factory for point-to-point messaging with the default messaging provider. This is intended more for backwards compatibility, as described in section 1.5 of the JMS 1.1 specification.

About this task

To configure a JMS queue connection factory for the default messaging provider, use the administrative console to complete the following steps. This task contains an optional step for you to create a new queue connection factory.

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure a queue connection factory.
3. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Queue connection factories**. This displays any existing JMS queue connection factories for the default messaging provider in the content pane.

5. Display the properties of the JMS connection factory. If you want to display an existing JMS connection factory, click one of the names listed.

   Alternatively, if you want to create a new JMS connection factory, click **New**, then specify the following required properties:

   Name Type the name by which the connection factory is known for administrative purposes.

   JNDI name
   Type the JNDI name that is used to bind the connection factory into the name space.

   Bus name
   Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics represent.
6. Review the other JMS queue connection factory [Settings], to check that the defaults are suitable. If needed, change the values according to your needs.

   If the connection factory is for use by client applications running outside of an application server, you need to specify suitable provider endpoints. For more information about configuring provider endpoints, see “Configuring a connection to a non-default bootstrap server” on page 1598.

   By default connections created using the connection factory in the server containers (for example, from an enterprise bean) are pooled using Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection pooling. You can modify the connection pool settings for the connection factory by selecting the Connection pool properties link in the Additional Properties section of the panel. For more information about changing the connection pool properties, see Changing connection pool settings with the wsadmin tool.
7. Click **OK**.
8. Save your changes to the master configuration.

Configuring a JMS topic connection factory for the default messaging provider

Use this task to configure a JMS topic connection factory for publish/subscribe messaging with the default messaging provider. This is intended more for backwards compatibility, as described in section 1.5 of the JMS 1.1 specification.

About this task

You need to configure JMS connection factories when deploying JMS applications that use publish/subscribe messaging.

To configure a JMS topic connection factory for the default messaging provider, use the administrative console to complete the following steps. This task contains an optional step for you to create a new topic connection factory.

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure a topic connection factory.
3. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Topic connection factories**. This displays any existing JMS topic connection factories for the default messaging provider in the content pane.
5. Optional: Display the properties of the JMS connection factory. If you want to display an existing JMS connection factory, click one of the names listed.

   Alternatively, if you want to create a new JMS connection factory, click **New**, then specify the following required properties:

   Name Type the name by which the connection factory is known for administrative purposes.

JNDI name
Type the JNDI name that is used to bind the connection factory into the name space.

Bus name
Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics represent.

6. Review the other properties for the connection factory, to check that the defaults are suitable. If needed, change the values according to your needs.

   If the connection factory is for use by client applications running outside of an application server, you need to specify suitable provider endpoints. For more information about configuring provider endpoints, see "Configuring a connection to a non-default bootstrap server" on page 1598.

   By default connections created using the connection factory in the server containers (for example, from an enterprise bean) are pooled using Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) connection pooling. You can modify the connection pool settings for the connection factory by selecting the Connection pool properties link in the Additional Properties section of the panel. For more information about changing the connection pool properties, see Changing connection pool settings with the wsadmin tool.
7. Click **OK**.
8. Save your changes to the master configuration.

Configuring a JMS queue for the default messaging provider

Use this task to configure a JMS queue for point-to-point messaging with the default messaging provider.

Before you begin

This task configures a JMS queue to use a queue destination on a service integration bus. The queue destination is the virtual location on the bus where messages are stored and processed for the JMS queue. If you have not created the required destination, use the task described in Configuring a destination for JMS queues.

About this task

To configure a JMS queue for the default messaging provider, use the administrative console to complete the following steps.

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure a queue.
3. Optional: Change the **Scope** check box to set the level at which the JMS queue is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Queues**. This displays any existing JMS queues for the default messaging provider in the content pane.
5. Display the properties of the JMS queue. If you want to display an existing JMS queue, click one of the names listed.

   Alternatively, if you want to create a new JMS queue, click **New**, then specify the following required properties:

   Name Type the name by which the queue is known for administrative purposes.

   JNDI name
   Type the JNDI name that is used to bind the queue into the name space.

   Queue name
   Select the name of the queue on a service integration bus that this JMS queue is to use.
6. Specify properties for the JMS queue, according to your needs.

7. Click **OK**.
8. Save your changes to the master configuration.

Configuring a JMS topic for the default messaging provider

Use this task to configure a JMS topic for publish/subscribe messaging with the default messaging provider.

Before you begin

This task configures a JMS topic to use a topic space destination on a service integration bus. The topic space (a hierarchical collection of topics) is the virtual location on the bus where messages are stored and processed for the JMS topic. If you have not created the required destination, use the task described in Configuring a destination for publish/subscribe messaging.

About this task

To configure a JMS topic for the default messaging provider, use the administrative console to complete the following steps. This task contains an optional step for you to create a new JMS topic.

1. Display the default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure a topic.
3. Optional: Change the **Scope** check box to set the level at which the topic is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Topics**. This displays any existing JMS topics for the default messaging provider in the content pane.
5. Display the properties of the JMS topic. If you want to display an existing JMS topic, click one of the names listed.

   Alternatively, if you want to create a new JMS topic, click **New**, then specify the following required properties:

   Name Type the name by which the topic is known for administrative purposes.

   JNDI name
   Type the JNDI name that is used to bind the topic into the name space.

   Topic name
   Type the name of the topic (as a qualifier in the topic space) that this JMS topic is to use.

   Topic space
   Type the name of the topic space that this JMS topic is to use.
6. Specify properties for the JMS topic, according to your needs.
7. Click **OK**.
8. Save your changes to the master configuration.

Configuring a JMS activation specification for the default messaging provider

A JMS activation specification is a configurable resource that enables a message-driven bean to communicate with the default messaging provider.

About this task

You create a JMS activation specification if you want to use a message-driven bean to communicate with the default messaging provider through Java EE Connector Architecture (JCA) 1.5. JCA provides Java connectivity between application servers such as WebSphere Application Server, and enterprise information systems. It provides a standardized way of integrating JMS providers with Java EE application servers, and provides a framework for exchanging data with enterprise systems, where data is transferred in the form of messages.

One or more message-driven beans can share a single JMS activation specification.

For WebSphere Application Server Version 5.1, the interface between message-driven beans and their destinations was the listener port. Although you cannot use listener ports with the default messaging provider in Version 7.0, you can continue to use listener ports with external JMS providers such as the WebSphere MQ messaging provider.

Listener ports can be manually started and stopped. When the listener port is stopped, the message-driven bean associated with it can no longer process messages. Furthermore, if a message-driven bean fails to process a message several times, the listener port is automatically stopped by the application server. Because a JMS activation specification is a group of messaging configuration properties not a component, it cannot be manually started and stopped. For this reason, to prevent a message-driven bean from processing messages you must complete the following tasks:

Stop the application that contains the message-driven bean.

Stop the messaging engine.

The following guidelines show which scenarios use activation specifications or listener ports:

- If you are using Java EE 1.4 and EJB 2.1 with the default message provider in WebSphere Application Server Version 7.0 or Version 6.x, you must use activation specifications. A connector message-driven bean uses JCA to access its resources, so the connector must be configured with an activation specification. This is for new bean development, and does not affect the conversion of message-driven beans from EJB 2.0 to EJB 2.1.
- If you are using Java EE 1.4 and EJB 2.1 with a third-party JMS provider in WebSphere Application Server Version 7.0 or Version 6.x, the decision depends on whether your JMS provider API is implemented with JCA. In Java EE 1.4, the JMS 1.1 API can be implemented with the JCA 1.5 API. If so, your message-driven bean is a JMS message-driven bean that is implemented as a connector message-driven bean, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for Java EE 1.3, and you must configure this EJB 2.1 message-driven bean in the same way as you would configure an EJB 2.0 message-driven bean, which in WebSphere Application Server is to use a listener port.
- If you are using Java EE 1.3 and EJB 2.0 with WebSphere Application Server Version 5.1, you must use listener ports. The message-driven beans are JMS message-driven beans that implement MessageListener, and there is no Java EE Connector Architecture (JCA) support. WebSphere Application Server Version 5.1 uses listener ports to associate message-driven bean classes with their JMS destinations.

All the activation specification configuration properties apart from **Name**, **JNDI name**, **Destination JNDI name**, and **Authentication alias** are overridden by appropriately named activation-configuration properties in the deployment descriptor of an associated EJB 2.1 message-driven bean. For an EJB 2.0 message-driven bean, the **Destination type**, **Subscription durability**, **Acknowledge mode** and **Message selector** properties are overridden by the corresponding elements in the deployment descriptor. For either type of bean the **Destination JNDI name** property can be overridden by a value specified in the message-driven bean bindings.

To configure a JMS activation specification for the default messaging provider, use the administrative console to complete the following steps. This task contains an optional step for creating a new JMS activation specification.

1. Display the default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure an activation specification.
3. Optional: Change the **Scope** check box to the scope level at which the activation specification is to be visible to applications, according to your needs.
4. In the content pane, under the Additional properties heading, click **Activation specifications**. This lists any existing JMS activation specifications for the default messaging provider in the content pane.

5. Display the properties of the JMS activation specification. If you want to display an existing activation specification, click one of the names listed.

   Alternatively, if you want to create a new activation specification, click **New**, then specify the following required properties:

   Name Type the name by which the activation specification is known for administrative purposes.

   JNDI name
   Type the JNDI name that is used to bind the activation specification into the JNDI name space.

   Destination type
   Whether the message-driven bean uses a queue or topic destination.

   Destination JNDI name
   Type the JNDI name that the message-driven bean uses to look up the JMS destination in the JNDI name space.

   Select the type of destination on the Destination type property.

   Bus name
   The name of the bus to connect to.

   Specify the name of the service integration bus to which connections are made. This must be the name of the bus on which the bus destination identified by the Destination JNDI name property is defined.

   You can either select an existing bus or type the name of another bus. If you type the name of a bus that does not exist, you must create and configure that bus before the activation specification can be used.
6. Specify properties for the JMS activation specification, according to your needs.
7. Optional: Specify the JMS activation specification connection properties that influence how the default messaging provider chooses the messaging engine to which your message-driven bean application connects. By default, the environment automatically connects applications to an available messaging engine on the bus. However you can specify extra configuration details to influence the connection process; for example to identify special bootstrap servers, or to limit connection to a subgroup of available messaging engines, or to improve availability or performance, or to ensure sequential processing of messages received. For information about why and how to do this, see How JMS applications connect to a messaging engine on a bus.
8. Click **OK**.
9. Save your changes to the master configuration.

Deleting JMS resources for the default messaging provider

Use this task with the WebSphere Application Server administrative console to delete JMS resources.

About this task

To delete JMS resources, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to delete resources.
3. In the content pane, under the Additional properties heading, select the link for the type of JMS resource:
   - Connection factory
   - Queue connection factory
   - Topic connection factory
   - Queue
   - Topic
   - Activation specification

This displays a list of the selected JMS resource type in the content pane.

4. Select the check box next to the JMS resource that you want to delete.
5. Click **Delete**
6. Save your changes to the master configuration.

Configuring JMS connection factory properties for durable subscriptions

Use this task to configure durable subscription properties of JMS connection factories for use by enterprise beans with the default messaging provider.

About this task

To enable applications to create durable subscriptions to JMS topics with the default messaging provider, you can set a number of properties on JMS connection factories.

If applications use message-driven beans to create durable subscriptions, you should set the properties on the JMS activation specification used by the message-driven beans, as described in Configuring JMS activation specifications for durable subscriptions, instead of using the task described in this topic.

This topic describes the setting of properties on a unified JMS connection factory. You can also set the same properties on a JMS topic connection factory instead.

To configure the durable subscription properties to a topic for use by enterprise beans with the default messaging provider, use the administrative console to complete the following steps:

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure connection factory properties.
3. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Connection factories**. This displays any existing JMS connection factories for the default messaging provider in the content pane.
5. Click the name of the connection factory you want to configure. This displays the properties for the connection factory in the content pane.
6. Specify the following properties for the connection factory:

   Client identifier
   This is the JMS client identifier that applications need to identify durable topic subscriptions created on all connections using this connection factory. For more information about client identifiers, see section 4.3.2 of the JMS 1.1 specification.

   Durable subscription home
   The name of the messaging engine used to store messages delivered to durable subscriptions for objects created from this JMS connection factory.

   This identifies the messaging engine where durable subscriptions are localized on the service integration bus. Administrators can manage the runtime state of durable subscriptions through publication points for that messaging engine.
7. Click **OK**.
8. Save your changes to the master configuration.

What to do next

When applications have created durable subscriptions, you can use the administrative console to manage the runtime state of those subscriptions, as described in “Administering durable subscriptions” on page 1251.

The JMS connection factory has some other advanced properties that you can configure to change the behavior for durable subscriptions. You should not normally need to change these properties from there defaults.

Read ahead
This controls read ahead optimization during message delivery. This defines whether or not the provider can stream messages to durable subscribers ahead of their requests (to provide a performance enhancement).

Share durable subscriptions
This controls whether or not durable subscriptions can be accessed simultaneously by several subscribers.

If you want to control read ahead optimization during message delivery for individual topics, you can set the Read ahead property on the topics.

Configuring JMS activation specification properties for durable subscriptions

Use this task to configure durable subscription properties of JMS activation specifications for use by message-driven beans with the default messaging provider.

About this task

To enable a message-driven bean (MDB) application to create durable subscriptions on JMS topics with the default messaging provider, you set a number of properties on the JMS activation specification used by the application.

If applications use message-driven beans to create durable subscriptions, you should set the properties on the JMS activation specification used by the message-driven beans, as described in this topic. Otherwise, for enterprise beans to create durable subscriptions, you should set the properties on the JMS connection factory as described in “Configuring JMS connection factory properties for durable subscriptions” on page 1594.

To configure the durable subscription properties to a topic for use by message-driven beans with the default messaging provider, use the administrative console to complete the following steps

1. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Select the default provider for which you want to configure activation specification properties.
3. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
4. In the content pane, under the Additional properties heading, click **Activation specifications**. This displays any existing JMS activation specifications for the default messaging provider in the content pane.
5. Click the name of the activation specification you want to configure. This displays the properties for the activation specification in the content pane.
6. Specify the following properties for the activation specification:

   Client identifier
   This is the JMS client identifier that applications need to identify durable topic subscriptions created on all connections using this activation specification. For more information about client identifiers, see section 4.3.2 of the JMS 1.1 specification.

   Durable subscription home
   The name of the messaging engine used to store messages delivered to durable subscriptions for objects created from this JMS activation specification. This is a required field when using a durable topic subscription.

This identifies the messaging engine where durable subscriptions are localized on the service integration bus. Administrators can manage the runtime state of durable subscriptions through publication points for that messaging engine.

Subscription durability
To be able to create durable subscriptions, set this property to Durable.

Subscription name
The subscription name needed for durable topic subscriptions. Required field when using a durable topic subscription.

Each JMS durable subscription is identified by a subscription name (specified on this property). A JMS connection also has an associated client identifier (specified on the Client identifier property), which is used to associate a connection and its objects with the list of messages (on the durable subscription) that is maintained by the JMS provider for the client.

This subscription name must be unique within a given client identifier.

7. Specify the properties for the activation specification, according to your needs.
8. Click **OK**.
9. Save your changes to the master configuration.

What to do next

When applications have created durable subscriptions, you can use the administrative console to manage the runtime state of those subscriptions, as described in “Administering durable subscriptions” on page 1251.

Enabling a provider to stream messages to cloned durable subscriptions

Use this task to enable a provider to stream messages to consumers ahead of their message requests. This is most often used by publish/subscribe consumers to provide a performance enhancement.

About this task

To indicate that a provider must stream messages to consumers, you can set the Read ahead property to Enabled. This property can be set on a connection factory to specify the behavior for all connections created using that connection factory. The property can also be set on JMS topics, to enable different behavior when sending messages to different JMS topics from the same connection.

You are recommended to leave this property set to Default, which enables the messaging provider to decide whether or not it should stream messages to consumers. The messaging provider makes this decision based on the environment in which the durable subscriber is running. You should only set this property to enable message streaming if you are sure that a durable subscription is used by only one consumer at a time.

- In a non-cloned environment, the default setting enables messaging streaming for durable subscribers.
- For cloned durable subscribers (that is, a durable subscriber that is part of an application cloned in a server cluster), the default setting prevents message streaming and the messages on the subscription are shared among the clones. If you are moving from a non-cloned environment, to cloned durable subscribers, you may see a drop in performance. If you are sure that a durable subscription is still used by only one consumer at a time, you can enable message streaming as described in this topic.

  Server clusters can be used as bus members only in WebSphere Application Server environments that support server clusters.

Messages that are streamed to the consumer but are not consumed before the consumer disconnects are unlocked when the consumer closes. Only then do those messages become available for consumption by other consumers.

To force the messaging provider to stream messages to cloned durable subscriptions, use the administrative console to complete the following steps to change the connection factory:

1. Display the JMS connection factory; for example, as described in “Configuring a unified JMS connection factory for the default messaging provider” on page 1586. All clones of a durable subscriber use the same JMS connection factory.
2. Set the Read ahead property to Enabled.
3. Click **OK**.
4. Save your changes to the master configuration.

Enabling CMP entity beans and messaging engine data stores to share database connections

Use this task to enable container-managed persistence (CMP) entity beans to share the database connections used by the data store of a messaging engine. This has been estimated as a potential performance improvement of 15% for overall message throughput, but can only be used for entity beans connected to the application server that contains the messaging engine.

About this task

To enable CMP entity beans to share the database connections used by the data store of a messaging engine, complete the following steps.

1. Configure the data store to use a data source that is not XA-capable. For more information about configuring a data store, see “Configuring a JDBC data source for a messaging engine” on page 1213.
2. Select the Share data source with CMP option.

   This option is provided on the JMS connection factory or JMS activation specification used to connect to the service integration bus that hosts the bus destination that is used to store and process messages for the CMP bean.

   For example, to select the option on a unified JMS connection factory, complete the following steps:

   a. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
   b. Select the default provider for which you want to configure a unified connection factory.
   c. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
   d. In the content pane, under Additional Properties, click **Connection factories**
   e. Optional: To create a new unified JMS connection factory, click **New**.

      Specify the following properties for the connection factory:

      Name Type the name by which the connection factory is known for administrative purposes.

      JNDI name
      Type the JNDI name that is used to bind the connection factory into the name space.

      Bus name
      Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics are assigned to.

   f. Optional: To change the properties of an existing connection factory, click one of the connection factories displayed. This displays the properties for the connection factory in the content pane.
   g. Select the check box for the Share data source with CMP field
   h. Click **OK**.
   i. Save your changes to the master configuration.

   The JMS connection factory can only be used to connect to a “local” messaging engine that is in the application server on which the CMP beans are deployed.

3. Deploy the CMP beans onto the application server that contains the messaging engine, and specify the same data source as used by the messaging engine. You can use the administrative consoles to complete the following steps:
   a. Optional: To determine the data source used by the messaging engine, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engines** → ***engine_name*** → **[Additional Properties] Message store** The **Data source name** field displays the name of the data source; by default:

      `jdbc/com.ibm.ws.sib/`*`engine_name`*

   b. Click **Applications** → **New Application** → **New Enterprise Application**.
   c. On the first Preparing for application install page, specify the full path name of the source application file (.ear file otherwise known as an EAR file), then click **Next**
   d. On the second Preparing for application install page, complete the following steps:
      1) Select the check box for the Generate Default Bindings property. Data source bindings (for EJB 1.1 JAR files) are generated based on the JNDI name, data source user name password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.
      2) Under Connection Factory Bindings, click the check box for the **Default connection factory bindings:** property, then type the JNDI name for the data source and optionally select a **Resource authorization** value.
      3) Click **Next**
4. If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for Step: Provide default data source mapping for modules containing 1.x entity beans, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.
5. If your application has CMP beans that are based on the EJB 1.x specification, for Step: Map data sources for all 1.x CMP, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error displays after you click **Finish** (step 13) and the installation is cancelled.
6. Complete other panels as needed.
7. On the Summary panel, verify the cell, node, and server onto which the application modules will install:
   a. Beside Cell/Node/Server, click the **Click here** link.
   b. Verify the settings on the Map modules to servers page displayed. Ensure that the application server that is specified contains the messaging engine and its data store.
   c. Specify the Web servers as targets that will serve as routers for requests to this application. This information is used to generate the plug-in configuration file (plugin-cfg.xml) for each Web server.
   d. Click **Finish**.

Results

For more information about installing applications, see Installing application files with the console.

Configuring a connection to a non-default bootstrap server

A bootstrap server is an application server running in the same cell, specifically the same core group, as the service integration bus.

About this task

Connection to a non-default bootstrap server is provided by a JMS connection factory or a JMS activation specification. The connection allows applications to use a bootstrap server with a non-default endpoint

address. The provider endpoint syntax example described here is also relevant to bootstrap endpoint configuration in other tasks, for example when configuring a service integration bus link.

To use JMS destinations of the default messaging provider, an application or MDB connects to a messaging engine on the target service integration bus where the destinations are assigned. For example, a JMS queue is assigned to a queue destination on a service integration bus.

Applications running in a server that is part of the same cell as the service integration bus can normally connect to a messaging engine on that bus without requiring the configuration of provider endpoints. If the cell has been split using two core groups, each defined with its own policies, client applications that are running in a client container and client applications running outside the WebSphere Application Server environment cannot automatically locate the required service integration bus so it is necessary to configure one or more provider endpoints, unless a core group bridge has been configured between the core groups in the same cell. Similarly, an application running on a server in one cell cannot connect to a bus in another cell without the configuration of provider endpoints, unless a core group bridge has been established between the two cells.

In the scenarios where provider endpoints are required, the clients or the servers in another bus must complete a bootstrap process through a bootstrap server. The bootstrap server does not have to be a member of the service integration bus, and it does not have to contain any messaging engines. For the application to locate the required bootstrap server, the provider endpoint property of the JMS connection factory or JMS activation specification used by the client application must be configured. When the bootstrap server receives the client request, it selects a messaging engine that matches the criteria specified by the connection factory or activation specification, for example the target transport chain, target group, connection proximity. It returns the location information for this messaging engine to the client, and the client transparently creates a new connection to the target messaging engine if necessary.

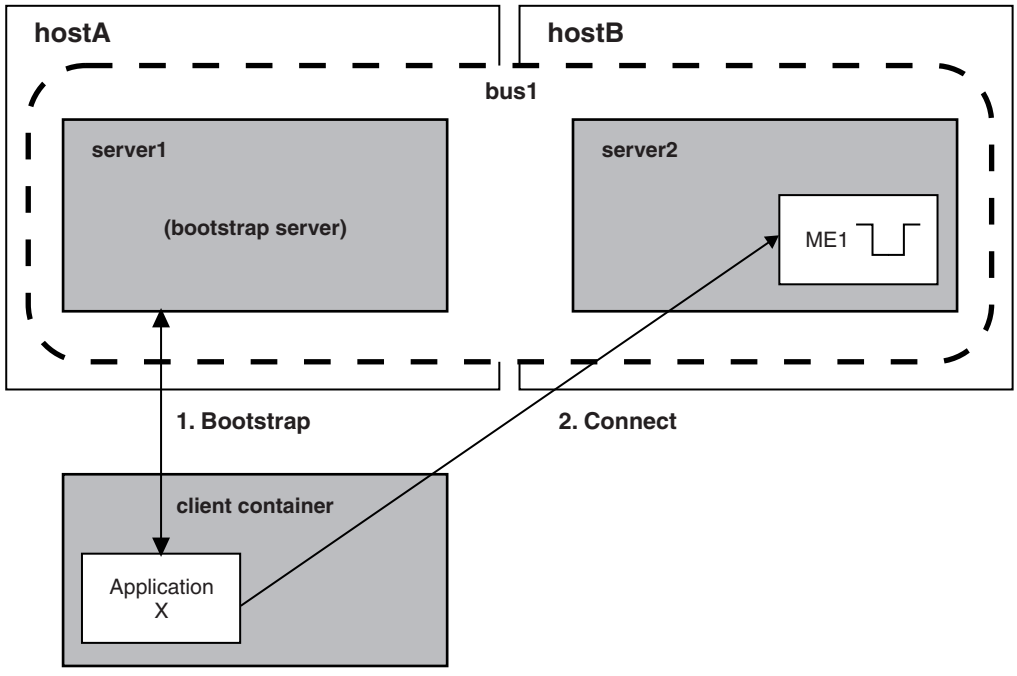


Figure 17. Connection to a messaging engine - Applications running outside an application server. This figure shows a client application running outside an application server. To connect to a messaging engine, the application connects first to a bootstrap server. The bootstrap server selects a messaging engine then tells the client application to connect to that messaging engine.

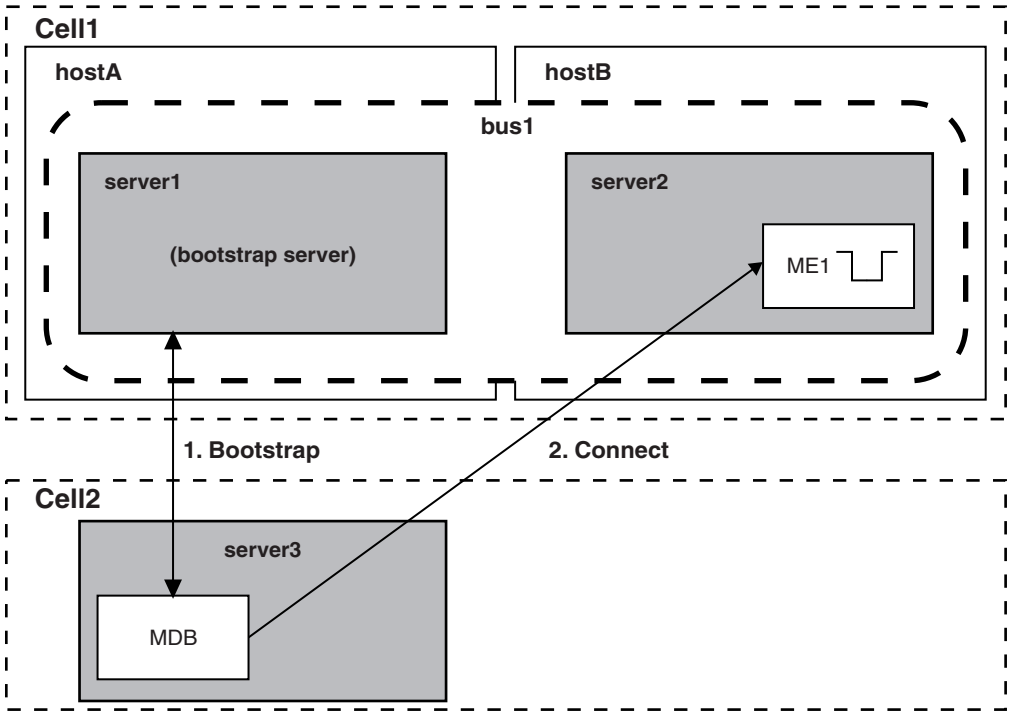


Figure 18. Connection to a messaging engine - MDB application connecting to a destination in a different cell. This figure shows an MDB running in an application server that is in a different cell to the bus that the MDB needs to be connected to in order to receive messages. To connect to a messaging engine, the MDB connects first to a bootstrap server. The bootstrap server selects a messaging engine then tells the MDB to connect to that messaging engine.

A bootstrap server listens on an endpoint that is defined by the combination of:

- the host name of the host on which the bootstrap server is running
- a specific port which is either SIB_END_POINT or, if security is enable, SIB_ENDPOINT_SECURE_ADDRESS
- a bootstrap transport chain

The properties of a JMS connection factory or activation specification used by an application control the selection of a suitable messaging engine and how the application connects to the selected messaging engine. If a provider endpoint is not specified, a default value is inserted depending on whether or not a password has been supplied. If the application does not supply a password, a default endpoint address of localhost:7276:BootstrapBasicMessaging is used. That is, by default, applications try to use a bootstrap server on the same host as the client, and using port 7276 and the predefined bootstrap transport chain called BootstrapBasicMessaging. If the application does supply a password, the default secure port of 7286 and transport chain of BootstrapSecureMessaging is provided to prevent transmission of an unencrypted password to the server.

Note: For the i5/OS platform, you must (at least) change the default host name from `localhost` to *`your.server.name`*.

If you want an application to use a bootstrap server with a different endpoint address, you need to specify the required endpoint address on the **Provider endpoints** property of the JMS connection factories or JMS activation specifications that the client application or MDB uses. You can specify one or more endpoint addresses of bootstrap servers using a comma separated list.

The endpoint addresses for bootstrap servers must be specified in every JMS connection factory that is used by applications outside of an application server. To avoid having to specify a long list of bootstrap

servers, you can provide a few highly-available servers as dedicated bootstrap servers. Then you only need to specify a short list of bootstrap servers on each connection factory.

This task is based on an application that uses a unified JMS connection factory. You can use the same task to configure a JMS queue connection factory or JMS topic connection factory, but during the task would need to select the appropriate type of connection factory instead of **JMS connection factory**. You can also use this task to configure a JMS activation specification instead of a JMS connection factory.

Note: When configuring a connection to a non-default bootstrap server, specify the required values using colons as separators. The syntax is as follows:

```
[ [host_name] [ ":" [ port_number] [ ":" chain_name] ] ]
```

Specifying host_name : chain_name instead of host_name : : chain_name (with two colons) is incorrect. The default value applies if you do not specify a value, but you must separate the fields with ″:″s.

For an application to use a bootstrap server with a non-default endpoint address, complete the following steps:

1. Identify the endpoint address of the application server that you want to use as the bootstrap server. The endpoint address has the form *host_name*:*port_number*:*chain_name* where:

   host_name
   is the name of the host on which the server runs. It can be an IP address. In the case of an IPv6 address, put square braces ([]) around *host_name*.The default is `localhost`.

   Note: For the i5/OS platform, you must (at least) change the default host name from `localhost` to *your.server.name*.

   port_number
   where specified, is one of the following addresses of the messaging engine hosting the remote end of the link:

   - SIB_ENDPOINT_ADDRESS if security is not enabled
   - For secure connections, SIB_ENDPOINT_SECURE_ADDRESS if security is enabled.

   This value is mandatory. The default is 7276 if the application has not specified a password, or 7286 if a password has been specified.

   To find either of these values by using the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Communications] Ports**.

   chain_name
   is the name of a predefined bootstrap transport chain used to connect to the bootstrap server. If not specified, the default is `BootstrapBasicMessaging` if a password has not been provided, or BootstrapSecureMessaging if a password has been provided.

   The following predefined bootstrap transport chains are provided:

   BootstrapBasicMessaging
   This corresponds to the server transport chain InboundBasicMessaging (JFAP-TCP/IP)

   BootstrapSecureMessaging
   This corresponds to the server transport chain InboundSecureMessaging (JFAP-SSL-TCP/IP)

   BootstrapTunneledMessaging
   Before you can use this bootstrap transport chain, you must define a corresponding server transport chain on the bootstrap server. (See **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engine inbound transports**.) This transport chain tunnels JFAP using HTTP wrappers.

BootstrapTunneledSecureMessaging
Before you can use this bootstrap transport chain, you must define a corresponding server transport chain on the bootstrap server. (See **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engine inbound transports**.) This transport chain tunnels JFAP using HTTP wrappers.

If you want to provide more than one bootstrap server, identify all the required endpoint addresses. Separate each endpoint address by a comma character. You should be able to specify the endpoint address for each bootstrap server; for example: for a server assigned non-secure port 7278, on host boothost1, and using the default transport chain BootstrapBasicMessaging:

```
boothost1:7278:BootstrapBasicMessaging
```

or

```
boothost1:7278
```

and for a server assigned secure port 7289, on host boothost2, and using the predefined transport chain BootstrapTunneledSecureMessaging:

```
boothost2:7289:BootstrapTunneledSecureMessaging
```

2. Optional: Configure the endpoint address of the bootstrap server on the Provider endpoint property of the connection factory.

   If the client application uses a JMS connection factory in the client container, use the Client Resource Configuration tool (ACRCT):

   a. Start the tool and open the EAR file for which you want to configure the JMS connection factory. The EAR file contents are displayed in a tree view.
   b. From the tree, select the JAR file in which you want to configure the JMS connection factory.
   c. Expand the JAR file to view its contents.
   d. Expand **Messaging Providers > Default Provider > Connection Factories**.
   e. Display the general properties of the connection factory:
      - If you want to use an existing JMS connection factory, click the name of the connection factory.
      - If you want to create a new JMS connection factory, click **New**.

        For more information about configuring a JMS connection factory in the JMS provider configuration for your application client, see Configuring new JMS providers with the Application Client Resource Configuration Tool.
   f. On the General tab, ensure that the **Provider Endpoints** property includes the provider endpoint address for each bootstrap server. Type the value as a comma-separated list of endpoint addresses; for example:

      ```
      boothost1:7278,boothost2:7289:BootstrapTunneledSecureMessaging
      ```
   g. Click **OK**.
   h. To save your changes, click **File > Save**.

   If the client application uses a JMS connection factory on the server, use the WebSphere Application Server administrative console:

   a. Start the WebSphere Application Server administrative console.
   b. To display the default messaging provider, click **Resources** → **JMS** → **JMS providers**.
   c. Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
   d. In the content pane click **Default messaging provider**. This displays a table of properties for the default messaging provider, including links to the types of JMS resources that it provides.
   e. In the content pane, under Additional Properties, click **Connection factories**. This displays any existing connection factories in the content pane.
   f. Display the general properties of the connection factory:

- If you want to use an existing JMS connection factory, click the name of the connection factory.
- If you want to create a new JMS connection factory, click **New**.

  For more information about configuring a JMS connection factory, see “Configuring a unified JMS connection factory for the default messaging provider” on page 1586.

g. Ensure that the **Provider Endpoints** property includes the provider endpoint address for each bootstrap server. Type the value as a comma-separated list of endpoint addresses; for example:

```
boothost1:7278,boothost2:7289:BootstrapTunneledSecureMessaging
```

h. Click **OK**.

i. Save your changes to the master configuration.

3. Optional: Follow this step to configure the endpoint address of the bootstrap server on the Provider endpoint property of the activation specification.

   If the client application uses a JMS activation specification on the server, use the WebSphere Application Server administrative console:

   a. Start the WebSphere Application Server administrative console.

   b. To display the default messaging provider, click **Resources** → **JMS** → **JMS providers**.

   c. Select the default provider for which you want to configure an activation specification.

   d. Optional: Change the **Scope** check box to the scope level at which the activation specification to be visible to applications, according to your needs.

   e. In the content pane, under the Additional Properties heading , click **Activation specifications**. This lists any existing JMS activation specifications for the default messaging provider in the content pane.

   f. Display the properties of the JMS activation specification:

   - If you want to use an existing JMS activation specification, click one of the names listed.
   - If you want to create a new JMS activation specification, click **New**.

     For more information about configuring a JMS activation specification, see “Configuring a JMS activation specification for the default messaging provider” on page 1591.

   g. Ensure that the **Provider Endpoints** property includes the provider endpoint address for each bootstrap server. Type the value as a comma-separated list of endpoint addresses; for example:

```
boothost1:7278,boothost2:7289:BootstrapTunneledSecureMessaging
```

   h. Click **OK**.

   i. Save your changes to the master configuration.

Configuring MDB throttling on the default messaging provider

Use this task to configure the throttling of messages for message-driven beans that you have deployed as JCA 1.5 resources on the default messaging provider.

About this task

Use this task if you want to throttle messages for a message-driven bean deployed as a J2EE Connector Architecture (JCA) 1.5 resource on the default messaging JMS provider.

The default messaging provider (the service integration bus JMS Resource Adapter) uses a special type of message throttling. You should use the throttling support described in this topic instead of the JCA 1.5 throttling of messages described in Throttling of inbound message flow for JCA 1.5 message-driven beans. You can leave the message-driven bean pools to the default size of 500.

The default messaging provider enables the throttling of message delivery to a message-driven bean through a configuration option on the JMS activation specification used to deploy the bean. This configuration option is labelled **Maximum concurrent endpoints** and can be found on the JMS activation specification panels in the administrative console.

- The maximum number of instances of each message-driven bean is controlled by the Maximum concurrent endpoint setting in the activation specification used to deploy the message-driven bean. This maximum concurrency limit helps prevent a temporary build up of messages from starting an excessive number of MDB instances. By default, the maximum number of concurrent MDB instances is set to 10.

  The Maximum concurrent endpoints field limits the number of endpoints (instances of a given message-driven bean) that process messages concurrently. If the maximum has been reached, new messages are not accepted from the messaging engine for delivery until an endpoint finishes its current processing.

  If the available message count (queue depth) associated with a message-driven bean is frequently high, and if your server can handle more concurrent work, you can benefit from increasing the maximum concurrency setting.

  If you set the maximum concurrency for a message-driven bean, be sure that you specify a value smaller than the maximum number of endpoint instances that can be created by the adapter that the message-driven bean is bound to. If necessary, increase the endpoint instance limit.
- An activation specification also has a **Maximum batch size** that refers to how many messages can be allocated to an endpoint in one batch for serial delivery. So, for example, if you have set the Maximum concurrent endpoints property to 10 and the Maximum batch Size property to 3, then there can be up to 10 endpoints each processing up to 3 messages giving a total of 30 messages allocated to that message-driven bean. If there are multiple message-driven beans deployed against a single activation specification then these maximum values apply to each message-driven bean individually.
- Take care to ensure that you always set the Maximum concurrent endpoints property is always less than the JCA pool size.

Note: You might want to tune the throttling of your message-driven beans, which is especially important on z/OS. Workload arriving on the destination the message-driven bean is consuming from might use up more server resource and therefore obstruct other activities. An example of this is when restarting MDB applications you find a backlog of messages. The number of messages can be throttled so that the message-driven bean can process them in the most efficient manner.

To configure the message throttling support of the default messaging provider (the service integration bus JMS Resource Adapter), use the administrative console to complete the following steps.

- Tune the maximum number of instances of a message-driven bean.

  The maximum concurrency is set in the activation specification used to deploy the message-driven bean.

  1. Click **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification_name*** → → **[Additional Properties] J2C activation specification custom properties**.
  2. View the maxConcurrency custom property. The default is value is 10. For high throughput primitive MDB tests, 40 was found to be an optimal value.
  3. Optional: To change the maxConcurrency setting, click the value field. This displays a panel for you to type a new value. In the Value field, type the new value then click **OK**. Save your changes to the master configuration.
- Tune the maximum batch size for a message-driven bean.

  By default, only a single message is delivered to a message-driven bean instance at one time. You can improve performance by batching messages to the message-driven bean. Each message-driven bean instance then receives a number (between 1 and the batch size) of messages at a time. A change in the maximum concurrency is likely to be beneficial if the available message count (queue depth) associated with the message-driven bean is frequently high. For more information about the available message count, see View the Available Message Count on a destination. The maximum batch size is set in the activation specification used to deploy the message-driven bean.

  1. Click **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification_name*** → → **[Additional Properties] J2C activation specification custom properties**.

2. View the maxBatchSize custom property. The default value is 1. For high throughput primitive MDB tests, 5 was found to be optimal value (providing a 20 per cent gain over batch size 1).
3. Optional: To change the maxBatchSize setting, click the value field. This displays a panel for you to type a new value. In the Value field, type the new value then click **OK**. Save your changes to the master configuration..

Protecting a message-driven bean from system resource problems

This task describes how to configure the system so that, in the event of a dependent external system resource problem, the enterprise application is stopped before messages are moved to an exception destination unnecessarily. This configuration also allows occasional problems with messages to be handled without blocking the enterprise application.

Before you begin

This task assumes that you have deployed an enterprise application containing a message-driven bean (MDB) that interacts with external system resources. To complete this task an exception destination must be configured for the messaging engine, and you need the following information:

- The enterprise application that contains the MDB
- The dependent external system resources
- An acceptable value for the **Sequential failed message threshold**, that is, the maximum number of sequential failures of delivery of messages, after which the MDB is stopped. This property applies to sets of messages.
- An acceptable value for the **Delay between failing message retries**, that is, the time in milliseconds before a failing message is available to be delivered to the MDB. Other messages may be delivered during this period, unless the**Sequential failed message threshold** and the maximum concurrency is set to 1.
- An acceptable value for the **Maximum failed deliveries per message**, that is, the maximum number of failed attempts to process a message, after which the message is forwarded from its intended destination to the exception destination. This property applies to individual messages.

About this task

When an MDB fails to process a message, the message is rolled back and made available to the MDB again. Typically, the messaging system is configured in one of the following ways:

1. Failed messages are retried a finite number of times, and if they continue to fail are moved to an exception destination allowing subsequent messages to be processed.
2. Failed messages are retried indefinitely until the problem is rectified.

Configuration (1) protects the MDB against an occasional problem message that prevents subsequent messages from being processed. However if there is a prolonged problem with a resource that the MDB's enterprise application is dependent on, for example a database, all messages that are sent to the destination may be moved to the exception destination.

Configuration (2) blocks the delivery of messages until the original failing message problem is resolved. This configuration prevents messages being moved to an exception destination unnecessarily, but it also blocks subsequent messages as soon as a single problem message fails to be processed.

An MDB's activation specification can be configured to stop an MDB endpoint automatically when it detects a number of failures with sequential messages, which indicates a problem with a dependent resource. When the problem is resolved, the MDB endpoint is restarted manually. This configuration tolerates occasional message failures, allowing individual problem messages to be moved to the exception destination without blocking the entire MDB.

Follow these steps to protect an enterprise application from dependent external system resource failures.

1. Navigate to the deployed enterprise application that contains the MDB.
2. From the MDB, navigate to its JMS activation specification, **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name***.
3. Enter a value for the **Sequential failed message threshold**.
4. Save the configuration and navigate to the destination to which the MDB is listening, **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***queue_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***.
5. Check **Override messaging engine blocked retry timeout default** and enter a value for the **Blocked retry timeout in milliseconds.**.
6. Enter a value for the **Maximum failed deliveries per message**.
7. Save your changes to the master configuration..

Results

You have configured the enterprise application to protect itself from the sort of external resource problem that can occur at any time. This means that, in the event of a system resource problem, the MDB is stopped automatically when the Sequential failed message threshold is reached for any message.

What to do next

When the system resource that failed becomes available, you can restart the system resource and resume the MDB. The messages that failed during the system resource downtime are retried instead of being left on an exception destination.

Example 1: Handling a planned outage of a message-driven bean's external resource:

This task describes how to configure the system so that, in the event of a dependent external system resource problem, the enterprise application can continue.

Before you begin

During the time that the system resource is unavailable, there must be no exceptions in the enterprise application or messages on the exception destination that need to be resolved later.

About this task

Add a maintenance level to an external system resource that is used by the deployed message-driven bean (MDB) of one of the company's enterprise applications. The act of applying the maintenance level requires the system resource (for example, a database) to be unavailable for about five minutes.

1. Navigate to the deployed enterprise application that contains the MDB.
2. From the MDB, navigate to its JMS activation specification, **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name*** and click **Pause** on the administrative panel for the MDB.
3. When you receive a JMX notification and a log entry indicating that the MDB has been paused, stop the database and apply the maintenance level. While the MDB is paused, no messages are sent to the exception destination and no error messages appear in the console related to the stopped database.
4. Restart the database and test that it is working as expected.
5. Log onto the administrative console again, navigate to the same enterprise application and click **Resume** on the administrative panel for the MDB. You can also resume the MDB using scripting using the JCA MBean. The initial JMX notification and log entry indicate which MBean to use to resume the MDB. The MDB begins to be driven with the messages that are on the destination.

Results

You have paused and resumed an application while an external resource that is uses is briefly not available.

Example 2: Automatically stopping a message-driven bean when a system resource becomes unavailable:

To prepare for a system resource becoming unavailable, configure the system to stop the message-driven bean automatically after a small number of message failures and alert you to the problem.

Before you begin

This task assumes that you have deployed an enterprise application containing a message-driven bean (MDB) that interacts with external system resources. To complete this task an exception destination must be configured for the messaging engine, and you need the following information:

- The enterprise application that contains the MDB
- The dependent external system resources
- Set a value of `3` for the **Sequential failed message threshold**. This is the maximum number of sequential failures of delivery of messages, after which the MDB is stopped. This property applies to sets of messages.
- Set a value of `5000` for the **Delay between failing message retries**, that is, the time in milliseconds before a failing message is available to be delivered to the MDB. Other messages may be delivered during this period, unless the**Sequential failed message threshold** and the maximum concurrency is set to `1`.
- Set a value of `5` for the **Maximum failed deliveries per message**, that is, the maximum number of failed attempts to process a message, after which the message is forwarded from its intended destination to the exception destination. This property applies to individual messages.

About this task

Scenario: The enterprise application is a continuously running system that uses a deployed MDB to access an external system resource.

When this resource experiences a problem and is no longer available, the deployed MDB can no longer access this resource and so the transaction associated with the MDB is rolled back and the message (msg1 in this scenario) is put back on the queue.

Instead of msg1 being made available to the MDB immediately, it is hidden for the MDB's **Delay between failing message retries** retry delay of five seconds.

The next message on the queue (msg2) is processed by the MDB. The external resource is still unavailable so the processing of this message also fails and the message's transaction is rolled back. This message is also hidden for five seconds and the next message on the queue (msg3) is processed, fails and is also hidden.

When the number of hidden messages reaches the **Sequential failed message threshold**, the MDB will not process any further messages until one of the hidden messages becomes re-available.

When the **Delay between failing message retries** for msg1 expires, msg1 is unhidden and reprocessed. Because the resource is still unavailable, it is re-hidden. This also happens to msg2 and msg3.

Note: A message is considered a failed message when it is rolled back one less than the **Maximum failed deliveries per message** limit (five times in this scenario). So once msg1 is unhidden for the forth time, rolled back and rehidden, the sequential failure count is incremented. At which point

msg2 becomes unhidden, rolled back and rehidden. Similarly, msg3 becomes unhidden, rolled back and rehidden. The sequential failure count reaches the **Sequential failed message threshold** and the MDB stops automatically. A JMX notification is emitted by the JCA MBean and a log entry alerts the system administrator that the MDB has stopped.

Note: In this scenario, the MDB is automatically stopped when the system resource has been unavailable for approximately 20 seconds. If the system resource was unavailable for less than this time, the system could continue normally, and without numerous messages being sent to the exception destination.

1. Navigate to the deployed enterprise application that contains the MDB.
2. From the MDB, navigate to its JMS activation specification, **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name***.
3. Enter a value of `3` for the **Sequential failed message threshold**.
4. Save the configuration and navigate to the destination to which the MDB is listening, **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***queue_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***.
5. Check **Override messaging engine blocked retry timeout default** and enter a value of `5000` for the **Blocked retry timeout in milliseconds.**.
6. Enter a value of `5` for the **Maximum failed deliveries per message**.
7. Save your changes to the master configuration..
8. When you receive a JMX notification and a log entry indicating that the MDB (or endpoint) has been paused, investigate the problem with the system resource that the MDB was using. While the MDB is paused, no messages are sent to the exception destination and no error messages appear in the console related to the stopped database.
9. When the system resource that failed becomes available, restart it.
10. Log onto the administrative console again, navigate to the same enterprise application and click **Resume** on the administrative panel for the MDB. You can also resume the MDB using scripting using the JCA MBean. The initial JMX notification and log entry indicate which MBean to use to resume the MDB. The MDB begins to be driven with the messages that are on the destination.

Results

You have configured the system to protect itself from external resource failures.

What to do next

When the MDB is resumed, the JCA MBean emits a JMX notification to indicate that the MDB has resumed. Messages on the queue are consumed, messages that had failed are retried, and the transaction commits.

Example 3: The system experiences problems with a problem message:

To prepare for a problem message, configure the system to move that message to an exception destination and allow other messages to be processed successfully.

Before you begin

This task assumes that you have deployed an enterprise application containing a message-driven bean (MDB) that interacts with external system resources. To complete this task an exception destination must be configured for the messaging engine, and you need the following information:

- The enterprise application that contains the MDB
- The dependent external system resources

- Set a value of 3 for the **Sequential failed message threshold**. This is the maximum number of sequential failures of delivery of messages, after which the MDB is stopped. This property applies to sets of messages.
- Set a value of 5000 for the **Delay between failing message retries**, that is, the time in milliseconds before a failing message is available to be delivered to the MDB. Other messages may be delivered during this period, unless the**Sequential failed message threshold** and the maximum concurrency is set to 1.
- Set a value of 5 for the **Maximum failed deliveries per message**, that is, the maximum number of failed attempts to process a message, after which the message is forwarded from its intended destination to the exception destination. This property applies to individual messages.

About this task

Scenario: The enterprise application is a continuously running system that uses a deployed MDB to access an external system resource.

When a problem message (msg1 in this scenario) is encountered, it is put back on the queue.

Instead of msg1 being made available to the MDB immediately, it is hidden for the MDB's **Delay between failing message retries** retry delay of five seconds.

The next message on the queue (msg2) is processed by the MDB. This message and subsequent messages succeeds.

When the **Delay between failing message retries** for msg1 expires, msg1 is unhidden and reprocessed. It is put back on the queue again.

The MDB continues to process subsequent messages normally but each time msg1 is processed, it is put back on the queue.

When the number of times msg1 has been unhidden, rolled back and rehidden reaches the **Maximum failed deliveries per message** limit (five times in this scenario), it is moved to the configured exception destination.

1. Navigate to the deployed enterprise application that contains the MDB.
2. From the MDB, navigate to its JMS activation specification, **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name***.
3. Enter a value of 3 for the **Sequential failed message threshold**.
4. Save the configuration and navigate to the destination to which the MDB is listening, **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***queue_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***.
5. Check **Override messaging engine blocked retry timeout default** and enter a value of 5000 for the Delay between failing message retries.
6. Enter a value of 5 for the **Maximum failed deliveries per message**.
7. Save your changes to the master configuration..

Results

You have configured the system to protect itself from external resource failures and send problem messages to the exception destination.

Example 4: Automatically stopping a message-driven bean when no exception destination has been defined:

To prepare for a system resource becoming unavailable or a problem message, configure the system to stop the message-driven bean automatically. To maintain message ordering, do not configure an exception destination.

Before you begin

This task assumes that you have deployed an enterprise application containing a message-driven bean (MDB) that interacts with external system resources. To complete this task no exception destination must be configured for the messaging engine, and you need the following information:

- The enterprise application that contains the MDB
- The dependent external system resources
- Set a value of `1` for the **Sequential failed message threshold**. This is the maximum number of sequential failures of delivery of messages, after which the MDB is stopped. This property applies to sets of messages.
- Set a value of `5000` for the **Delay between failing message retries**, that is, the time in milliseconds before a failing message is available to be delivered to the MDB. Other messages may be delivered during this period, unless the**Sequential failed message threshold** and the maximum concurrency is set to `1`.
- An acceptable value for the **Maximum failed deliveries per message**, that is, the maximum number of failed attempts to process a message, after which the message is forwarded from its intended destination to the exception destination. This property applies to individual messages.

About this task

Scenario: The enterprise application is a continuously running system that uses a deployed MDB to access an external system resource.

When a problem message (msg1 in this scenario) is encountered, it is put back on the queue.

Instead of msg1 being made available to the MDB immediately, it is hidden for the MDB's **Blocked retry timeout in milliseconds** retry delay of 30 seconds.

When the number of hidden messages reaches the **Sequential failed message threshold**, the MDB will not process any further messages until one of the hidden messages becomes re-available. In this scenario, this threshold is reached as soon as msg1 is hidden.

When the **Blocked retry timeout in milliseconds** for msg1 expires, msg1 is unhidden and reprocessed.

This process is repeated until msg1 reaches its **Maximum failed deliveries per message** limit (five times in this scenario).

Once msg1 is unhidden for the forth time, rolled back and rehidden, the **Sequential failed message threshold** is reached and the MDB stops automatically. A JMX notification is emitted by the JCA MBean and a log entry alerts the system administrator that the MDB has stopped.

1. Navigate to the deployed enterprise application that contains the MDB.
2. From the MDB, navigate to its JMS activation specification, **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name***.
3. Enter a value of `1` for the **Sequential failed message threshold**.
4. Save the configuration and navigate to the destination to which the MDB is listening, **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***queue_name*** or **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations** → ***topic_space_name***.

5. Check **Override messaging engine blocked retry timeout default** and enter a value of 30 for the **Blocked retry timeout in milliseconds.**.
6. Enter a value of 5 for the **Maximum failed deliveries per message**.
7. Save your changes to the master configuration..
8. When you receive a JMX notification and a log entry indicating that the MDB (or endpoint) has been paused, investigate the problem. While the MDB is paused, as no exception destination is configured, msg1 remains on the queue. No other messages are processed
9. If you resume the MDB but the problem with the failing message continues, the maximum failed deliveries limit is reached on the first retry of the message, but as no exception destination is configured the message is not moved to another queue. Instead, the whole queue point is blocked to all consumers for the destination block retry interval (30 seconds). After this time, consumers begin again. If the failing message is still there, and fails again, the queue point is blocked for another 30 seconds. This continues until you remove the failing message from the queue, either by deleting it manually or solving the problem with it, and in doing so allowing the consuming application to succeed in processing
10. Log onto the administrative console again, navigate to the same enterprise application and click **Resume** on the administrative panel for the MDB. You can also resume the MDB using scripting using the JCA MBean. The initial JMX notification and log entry indicate which MBean to use to resume the MDB. The MDB begins to be driven with the messages that are on the destination.

Results

You have configured the system to protect itself from external resource failures while maintaining message ordering.

What to do next

When the MDB is resumed, the JCA MBean emits a JMX notification to indicate that the MDB has resumed. Messages on the queue are consumed, messages that had failed are retried, and the transaction commits.

Sample JMS 1.1 application client

This topic provides a typical example of JMS 1.1 application client code.

About this task

```
import java.util.Hashtable;
import javax.jms.JMSException;
import javax.naming.Context;
import javax.naming.*;
import javax.jms.*;


public class JMSppSampleClient
{
  public static void main(String[] args)
  throws JMSException, Exception

  {
    String  messageID                = null;
    String  outString                = null;
    String  cfName                   = "jms/blueconfactory";
    String  qnameIn                  = "java:comp/env/jms/Q1";
    String  qnameOut                 = "jms/bluequename";
    boolean verbose                  = false;

    Session               session    = null;
    Connection            connection = null;
    ConnectionFactory     cf         = null;
```

```
  MessageProducer        mp         = null;
  Destination            destination = null;

  try {

   Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.ibm.websphere.naming.WsnInitialContextFactory");
   env.put(Context.PROVIDER_URL, "iiop://localhost:2809");
   Context initialContext = new InitialContext(env);
System.out.println("Getting Connection Factory");

cf= (ConnectionFactory)initialContext.lookup( cfName );

System.out.println("Getting Queue");
destination =(Destination)initialContext.lookup(qnameOut);

      System.out.println("Getting Connection for Queue");
connection = cf.createConnection();

      System.out.println("staring the connection");
connection.start();

      System.out.println("creating session");
session = connection.createSession(false, 1);

      System.out.println("creating messageProducer");
mp = session.createProducer(destination);

      System.out.println("creating TextMessage");
TextMessage outMessage = session.createTextMessage("this is test application");

      System.out.println("sending Message");
mp.send(outMessage);

mp.close();
session.close();
connection.close();
  }
  catch (Exception je)     {}
```

Interoperating with a WebSphere MQ network

The default messaging provider (service integration) can interoperate with a WebSphere MQ network by using a WebSphere MQ link or a WebSphere MQ server. Alternatively, you can use WebSphere MQ as your messaging provider. Each type of connectivity is designed for different situations, and provides different advantages. Choose the most appropriate interoperation method for each of your messaging applications.

About this task

WebSphere Application Server can interoperate with WebSphere MQ in the following ways:

- By configuring WebSphere MQ as an external JMS provider using the WebSphere MQ messaging provider.
- By connecting a service integration bus to a WebSphere MQ network using the default messaging provider and a WebSphere MQ link.
- By integrating WebSphere MQ queues into a bus using the default messaging provider and a WebSphere MQ server.

A WebSphere MQ link provides a traditional WebSphere MQ -style solution to connecting resources. A WebSphere MQ server adds the ability to directly access WebSphere MQ queues from within a bus.

Table 15. The different ways of interoperating with WebSphere MQ

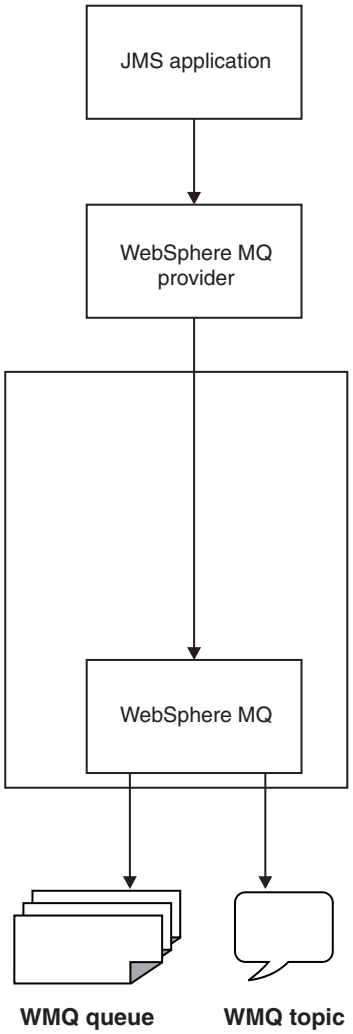
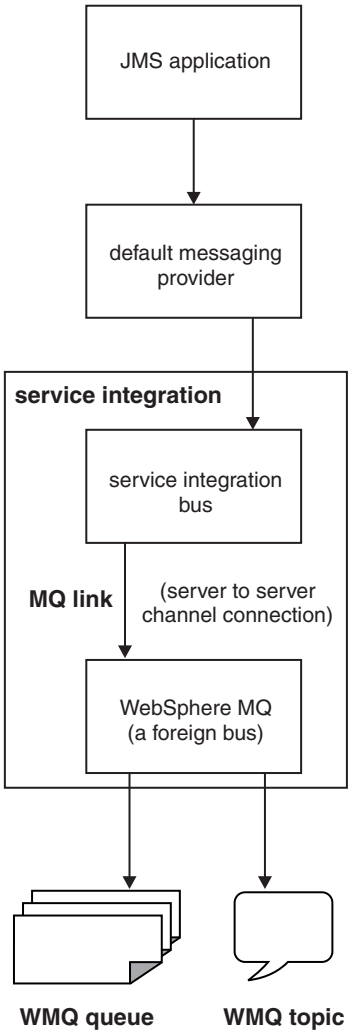
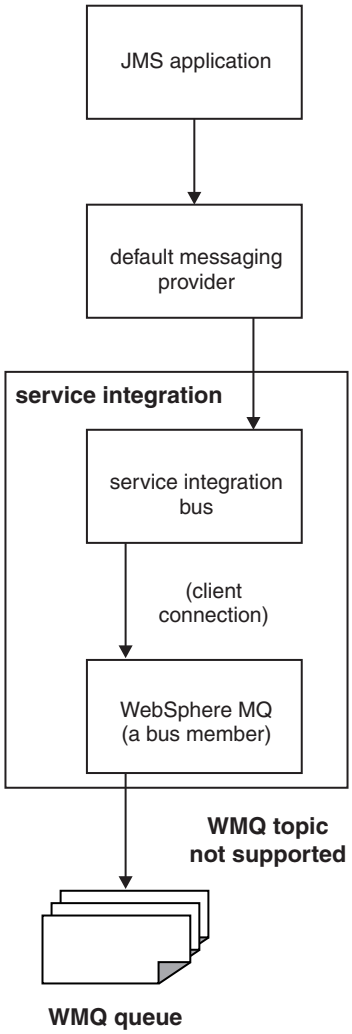
No bus: The WebSphere MQ messaging provider	WebSphere MQ network as a foreign bus (using a WebSphere MQ link)	WebSphere MQ queue manager or queue-sharing group as a bus member (using a WebSphere MQ server)
		

Table 15. The different ways of interoperating with WebSphere MQ (continued)

No bus: The WebSphere MQ messaging provider	WebSphere MQ network as a foreign bus (using a WebSphere MQ link)	WebSphere MQ queue manager or queue-sharing group as a bus member (using a WebSphere MQ server)
The WebSphere MQ messaging provider does not use service integration. It provides JMS messaging access to WebSphere MQ from WebSphere Application Server.	A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network.When you use a WebSphere MQ link, the messaging bus is seen by the WebSphere MQ network as a virtual queue manager, and the WebSphere MQ network is seen by service integration as a foreign bus. A WebSphere MQ link allows WebSphere Application Server applications to send point-to-point messages to WebSphere MQ queues (defined as destinations in the service integration bus), and allows WebSphere MQ applications to send point-to-point messages to destinations in the service integration bus (defined as remote queues in WebSphere MQ). The link also allows WebSphere Application Server applications to subscribe to messages published by WebSphere MQ applications, and WebSphere MQ applications to subscribe to messages published by WebSphere Application Server applications. The link ensures that messages are converted between the formats used by WebSphere Application Server and those used by WebSphere MQ.	A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. For interoperation with WebSphere Application Server Version 7.0, the version of WebSphere MQ must be WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ (distributed platforms) Version 7 or later. A WebSphere MQ server supports the high availability and optimum load-balancing characteristics provided by a WebSphere MQ for z/OS network. A WebSphere MQ server defines the connection and quality of service properties used for the connection, and also ensures that messages are converted between the formats used by WebSphere Application Server and those used by WebSphere MQ.

For more information about these approaches, see Learning about interoperating with a WebSphere MQ network.

To interoperate with a WebSphere MQ network complete one or more of the following steps.

- Choose the most appropriate interoperation method for each of your messaging applications. Complete this step if your existing or planned messaging environment involves both WebSphere MQ and WebSphere Application Server systems, and it is not clear to you whether you should use the default messaging provider, the WebSphere MQ messaging provider, or a mixture of the two.
- Configure WebSphere MQ as an external JMS provider (WebSphere MQ messaging provider).
- Use WebSphere MQ links to connect a bus to a WebSphere MQ network.
- Use WebSphere MQ server to integrate WebSphere MQ queues into a bus.

Using WebSphere MQ links to connect a bus to a WebSphere MQ network

If you operate within a WebSphere Application Server environment, sending messages across a service integration bus, you can also exchange point-to-point and publish/subscribe messages with applications in a WebSphere MQ network. To do this, you configure a foreign bus connection that links to a WebSphere MQ network through a WebSphere MQ link.

Before you begin

Decide on which method you want to use to configure these resources. You can configure a WebSphere MQ link by using the administrative console as described in this task, or you can configure a WebSphere MQ link by using the wsadmin tool.

About this task

A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network. The link operates on a messaging engine in a service integration bus to provide functions that simplify and automate interoperation with WebSphere MQ.

Using the WebSphere MQ link panels of the WebSphere Application Server administration console, you can choose:

- The WebSphere MQ queue manager or queue-sharing group in the WebSphere MQ network through which your messages will flow
- Whether to enable WebSphere Application Server applications to publish and subscribe to a message broker in the WebSphere MQ network
- Learn about interoperating with a WebSphere MQ network.
- Create a new WebSphere MQ link.
- Administer an existing WebSphere MQ link.
- Create applications that can interoperate with WebSphere MQ. For more information, see Learning about programming for interoperation with WebSphere MQ.

Creating a new WebSphere MQ link:

A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network. Use the foreign bus connection wizard to create a foreign bus and link it to the WebSphere MQ network through a WebSphere MQ link.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new WebSphere MQ link by using the administrative console as described in this task, or you can create a new WebSphere MQ link by using the wsadmin tool.

The following resources must be defined in WebSphere Application Server:

- A service integration bus that you want to connect from (known as the local bus) with at least one bus member.

The following resources must be defined in WebSphere MQ:

- A queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network.
- A listener that is configured and running.
- (optionally) A sender channel to receive messages on the local bus, a receiver channel to send messages from the local bus, or both.
- (For publish-subscribe messaging) A topic and input queue for broker publish-subscribe flow.

About this task

You use the foreign bus connection wizard to connect a bus and a WebSphere MQ network. You can configure the connection for either point-to-point or publish-subscribe messaging.

Specifically, the wizard helps you configure the following resources:

- The bus and messaging engine on which the WebSphere MQ link is defined.
- The foreign bus that represents the WebSphere MQ network.
- The WebSphere MQ link.

- (Optionally) The sender and receiver channels and protocol. Using the wizard you can choose to define no channels (and add them afterward), or one channel if your WebSphere MQ link is to be one-way, or both channels.
- (Optionally) Security for messages flowing across the link.
- (Optionally) A publish/subscribe broker profile and associated topic mappings, to allow publication and subscription with a broker in the WebSphere MQ network.

The wizard does not ask you to set all possible properties of a WebSphere MQ link, and many of the properties are set to default values. You can fine tune these properties afterward if necessary, by modifying the WebSphere MQ link.

For a sample configuration showing a systems view of the setup for a WebSphere MQ link, see "WebSphere MQ link sample configuration" on page 1617.

1. Use the foreign bus connection wizard to connect a bus and a WebSphere MQ network to use point-to-point messaging or publish-subscribe messaging.

   A WebSphere MQ link is created and configured as part of the action of the wizard.

   Note: You can subsequently convert a point-to-point connection to a publish/subscribe connection, by adding a publish/subscribe broker on the WebSphere MQ link for the connection.
2. Optional: Modify the new WebSphere MQ link.

   When you create a new WebSphere MQ link, the following properties are set to default values:
   - Description
   - Adoptable
   - Exception destination
   - Initial state
   - Nonpersistent message speed

   You can fine tune these properties by modifying the link.
3. Optional: Add or modify the WebSphere MQ receiver channel.

   If you did not choose to **Enable Service integration bus to WebSphere MQ message flow** in the foreign bus connection wizard, you have not yet defined a WebSphere MQ receiver channel. If you did choose this option in the wizard, you have defined the receiver channel name, host name and communication port, and the wizard has used default values for the following properties:
   - Inbound nonpersistent message reliability
   - Inbound persistent message reliability
   - Prefer queue points local to this link's messaging engine
   - Initial state

   You can fine tune these properties by modifying the channel.
4. Optional: Add or modify the WebSphere MQ sender channel

   If you did not choose to **Enable WebSphere MQ to Service integration bus message flow** in the foreign bus connection wizard, you have not yet defined a WebSphere MQ sender channel. If you did choose this option in the wizard, you have defined the sender channel name, host name, communication port and transport chain, and the wizard has used default values for the following properties:
   - Disconnect interval
   - Short retry count
   - Short retry interval
   - Long retry count
   - Long retry interval
   - Initial state

You can fine tune these properties by modifying the channel.

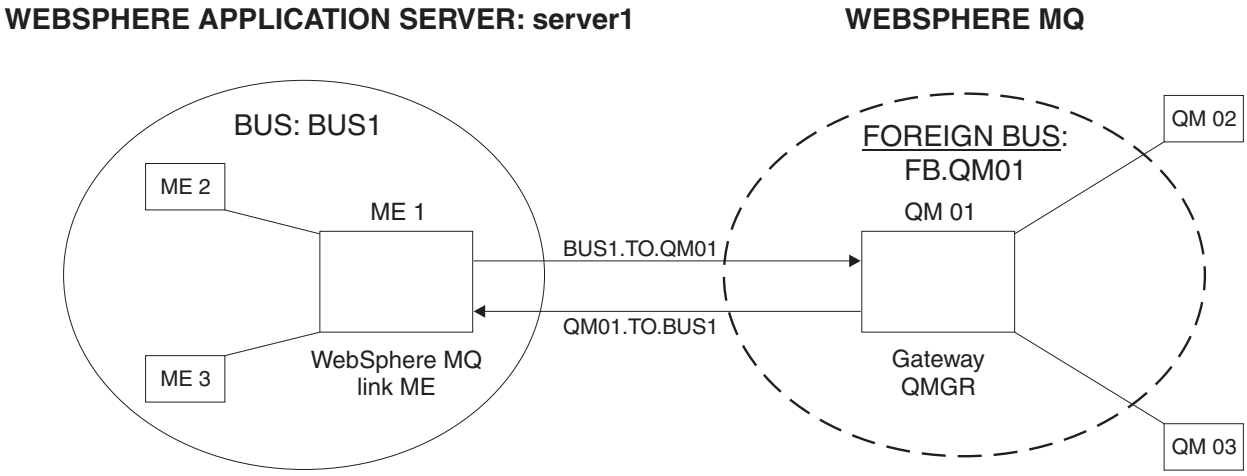
WebSphere MQ link sample configuration:

This topic describes a sample configuration sequence for connecting a WebSphere MQ link messaging engine on a WebSphere Application Server to a WebSphere MQ queue manager or queue-sharing group (known as a “gateway queue manager”) in a WebSphere MQ network.

Sample configuration context

This sample shows a set up required to achieve connectivity between a WebSphere Application Server messaging engine (ME), and a WebSphere MQ (Version 5.3 and above) network. Connectivity is achieved using WebSphere MQ link definitions (on a WebSphere Application Server) and WebSphere MQ channels (on a WebSphere MQ network). A WebSphere MQ link definition can be thought of as synonymous with a WebSphere MQ channel definition. A WebSphere MQ link defines the properties of both the sending and receiving ends of the link on a WebSphere Application Server.

The figure below shows an overview of the sample configuration.



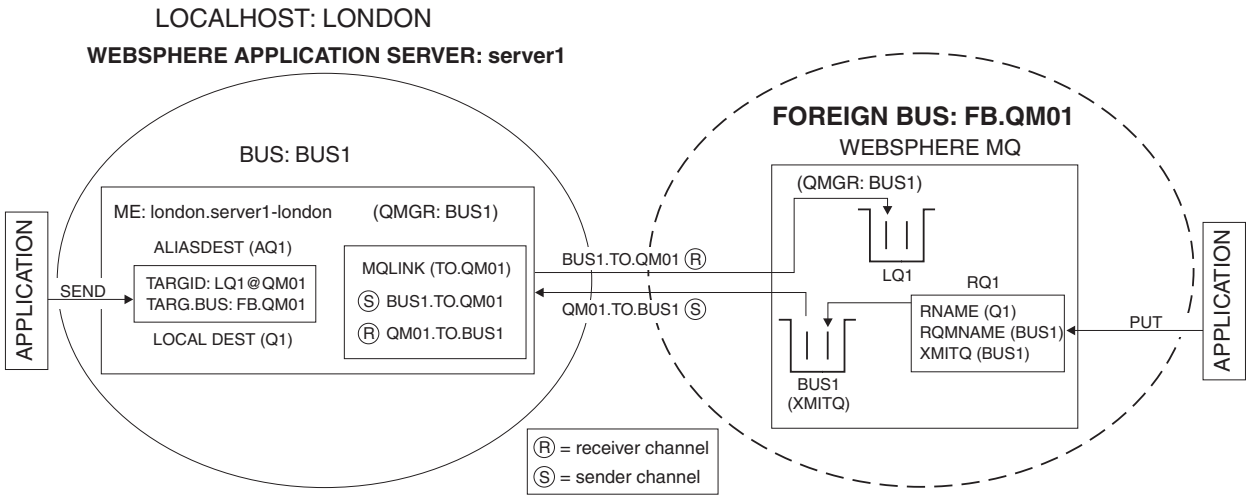
Configuration

- **Software**: IBM WebSphere Application Server and IBM WebSphere MQ.
- **Machine name**: LONDON (this is the local host).
- **WebSphere Application Server bus names**: Local Bus Name: BUS1. Foreign Bus Name: QM01 (this is the virtual bus name by which a remote WebSphere MQ queue manager is known to WebSphere Application Server; the remote WebSphere MQ queue manager may in turn be a part of a network of WebSphere MQ queue managers).
- **Name of WebSphere Application Server**: server1.
- **WebSphere Application Server messaging engine name**: london.server1-london.
- **WebSphere Application Server messaging engine virtual queue manager name**: BUS1 (this is the name by which the WebSphere Application Server messaging engine is known to a network of WebSphere MQ queue managers). As there is one messaging engine in the bus that is acting as a gateway to the WebSphere MQ network and, therefore, is used to send messages from, or receive messages for, any messaging engine in the bus, it is given the bus name.

- **WebSphere MQ queue manager name**: QM01 (This is the name of the gateway queue manager to the WebSphere MQ network. If the WebSphere MQ network consists only of this single queue manager, then it is also the name of the target queue manager. Otherwise, messages may first be routed to the gateway queue manager, from where they could be directed to the target queue manager.)
- **Listeners**:
  - By default, on a WebSphere Application Server, the listener for (unsecured) inbound connections from a WebSphere MQ JMS client, or from a WebSphere MQ queue manager, listens on port 5558.
  - By default, a listener is started on WebSphere MQ to listen on port 1414.

Overview of resource properties

The figure below shows the resources and their properties.



Properties of resources defined in WebSphere Application Server

- **WebSphere MQ link (a WebSphere MQ link is defined at the messaging engine level)**:
  - **Name**: (TO.QM01).
  - **Description**: (Link to send and receive data to and from queue manager QM01).
  - **Foreign bus name**: (FB.QM01).
  - **Queue manager name**: (BUS1) - the virtual queue manager name by which WebSphere Application Server is known to WebSphere MQ.
  - **Batch Size**: 50 (default).
  - **Maximum message size**: 4194304 (default).
  - **Heartbeat**: 300 (default).
  - **Sequence wrap**: 999999999 (default).
  - **Nonpersistent message speed**: Fast (default).
  - **Adoptable**: *Ticked* (default).
  - **Initial state**: Started (default).
- **WebSphere MQ link receiver (this is defined at the messaging level when defining the WebSphere MQ link):**
  - **Receiver channel name**: (QM01.TO.BUS1).
  - **Inbound nonpersistent message reliability**: Reliable (default).
  - **Inbound persistent message reliability**: Assured (default).
  - **Initial State**: Started (default).

- **WebSphere MQ link sender (this is defined at the messaging level when defining the WebSphere MQ link):**
  - **Sender channel name**: (BUS1.TO.QM01).
  - **Host name**: (LONDON).
  - **Port**: 1414 (default).
  - **Disconnect interval**: 900 (default).
  - **Short retry count**: 10 (default).
  - **Short retry interval**: 60 (default).
  - **Long retry count**: 999999999 (default).
  - **Long retry interval**: 1200 (default).
  - **Initial State**: Started (default).
- **Server**: By default server1 is created during installation (this has been used for this example).
- **Local Bus**: When installing an application server, a default bus is created. The name for this is produced from the node name specified at install time. It is london in this example setup.
- **Foreign Bus**: (This is defined at the bus level and needs to be specified on the WebSphere MQ link definition and also on alias destinations.)
  - **Name**: (FB.QM01).
  - **Description**: (Foreign bus needed to send messages to queue manager QM01).
- **Destination defaults on foreign bus:**
  - **Default priority**: (0).
  - **Reliability**: Assured persistent (default).
  - **Maximum reliability**: Assured persistent (default).
  - **Send allowed**: *Ticked*.

  Note: In this example context security is not set.
- **Routing definition on foreign bus:**
  - **Routing type**: MQLink.
  - **Name**: (RD.QM01).

    Routing definition can be created from the foreign bus collection panel which you navigate to through **Buses**>**busname**. The name you supply is for administration purposes only, it has no effect on routing capacity or operation.
  - **Inbound userid**: In this example this field is left set to blanks.
  - **Outbound userid**: In this example this field is left set to blanks.

Properties of resources defined in WebSphere MQ

These examples use the syntax of the runmqsc definitional program for WebSphere MQ.

1. **Sender Channel**

```
DEFINE    CHL(QM01.TO.BUS1)
          CHLTYPE(SDR)
          TRPTYPE(TCP)
          CONNAME('LONDON(5558)')
          XMITQ(BUS1)
```

   Define a TCP sender channel called 'QM01.TO.BUS1' with transmission queue 'BUS1' to WebSphere MQ.

   Let all other channel properties default.

2. **Receiver Channel**

```
DEFINE    CHL(BUS1.TO.QM01)
          CHLTYPE(RCVR)
          TRPTYPE(TCP)
```

Define a TCP receiver channel called 'BUS1.TO.QM01'.

Let all other channel properties default.

3. **Transmission Queue**

```
DEFINE   QL(BUS1)
         USAGE(XMITQ)
```

Define a local queue called 'BUS1' which will be used as the transmission queue for the sender channel.

Let all other channel properties default.

Sending messages from WebSphere MQ to a WebSphere Application Server destination

To send messages from WebSphere MQ to WebSphere Application Server, the following resources are defined:

- Resources on WebSphere MQ
  1. Remote Queue

```
DEFINE   QR(RQ1)
         RNAME(Q1)
         RQMNAME(BUS1)
         XMITQ(BUS1)
```

     Define a remote queue definition called 'RQ1'. This definition specifies that the remote queue name is 'Q1' and the remote queue manager name is 'BUS1'. The transmission queue associated with the remote queue definition is 'BUS1'.

     Let all other channel properties default.
- Resources on WebSphere Application Server
  1. Create a queue destination on the messaging engine with the MQ link (this is defined at the bus level):
     - Identifier (Q1).
     - Description: (Local queue to receive messages from a WebSphere MQ network).
     - Reliability: Assured persistent (default).
  2. Create default messaging resources. The JMS queue points to the bus queue Q1:

     Let this default to the local bus and server1 (london:server1).
  3. An application sending messages would have WebSphere MQ resources configured for the queue manager QM01 and the WebSphere MQ queue destination would use the RQ1 queue.

An application connected to queue manager QM01 would put messages to remote queue RQ1. These messages would be routed to Q1 on the target messaging engine.

Sending messages from WebSphere Application Server to a target local queue on WebSphere MQ

To send messages from WebSphere Application Server to WebSphere MQ, the following resources are defined:

- Resources on WAS: Alias Destination (this is defined at the bus level):
  - Identifier(AQ1).
  - Description(Alias for target queue LQ1).
  - Bus( ) - leave this blank to let it default.
  - Target Identifier(LQ1@QM01).
  - Target Bus(FB.QM01).
  - Reliability(Inherit) - default.
  - Maximum Reliability(Inherit) - default.
  - Producer can override quality of service(Inherit) - default.

  - Send allowed(Inherit) - default.
  - Receive allowed(Inherit) - default.
  - Default priority(-1) - default.
  - Reply destination( ) - default.
  - Reply destination bus( ) - default.
  - Default Forward Routing Path( ) - default.
- Resources on WebSphere MQ: Local Queue

  ```
  DEFINE   QL(LQ1)
  ```

  Let all other channel properties default.

An application connected to server1 would put messages to alias destination AQ1. These messages would be routed to LQ1 on target queue manager QM01.

Administering an existing WebSphere MQ link:

The WebSphere MQ link enables the exchange of point-to-point and publish/subscribe messages with a WebSphere MQ network. After you have created a WebSphere MQ link you can perform various administrative actions on the link.

About this task

The WebSphere MQ link connects a WebSphere Application Server and a WebSphere MQ network. Because these two systems are working together, their functions need to be in step so that each is aware of the status of the other. Administrative actions on the WebSphere MQ link and its functions include the following:

- Modify individual functions of the WebSphere MQ link .

  You can modify the link itself, add or modify the sender channel and receiver channel, or define a broker profile and associated topic mappings.
- Modify security for a WebSphere MQ link.
- View the status of a WebSphere MQ link and its components. You can view the status of a WebSphere MQ link and its sender and receiver channels, and you can view the status of subscriptions for a WebSphere MQ link publish/subscribe broker profile.
- Start a WebSphere MQ link.
- Stop a WebSphere MQ link.

  When you stop the link, all its functions are stopped too. For example, stopping a WebSphere MQ link with broker profiles on it might leave a message broker in the WebSphere MQ network with a backlog of messages. For more information, see “Stopping a WebSphere MQ link” on page 1633, “Stopping the sender channel on a WebSphere MQ link” on page 1635 and “Stopping the receiver channel on a WebSphere MQ link” on page 1634.
- Delete a WebSphere MQ link or one of its components. When you remove a foreign bus connection between a service integration bus and a WebSphere MQ network, you also delete the associated WebSphere MQ link along with any publish/subscribe broker profiles and topic mappings. You can also delete a WebSphere MQ link publish/subscribe broker profile or delete a topic mapping from a WebSphere MQ link.

Modifying a WebSphere MQ link:

How and when to modify the properties of a WebSphere MQ link.

Before you begin

Decide on which method you want to use to configure these resources. You can modify a WebSphere MQ link by using the administrative console as described in this task, or you can modify a WebSphere MQ link by using the wsadmin tool.

You need to know the name of the bus, and the messaging engine on the bus that contains the WebSphere MQ link you want to modify.

About this task

A WebSphere MQ link provides a server to server channel connection between a service integration bus and a WebSphere MQ queue manager or queue-sharing group, which acts as the gateway to the WebSphere MQ network. At any time after you create a new WebSphere MQ link, you can modify its properties.

When you use the foreign bus connection wizard to connect a bus and a WebSphere MQ queue manager or queue-sharing group (known as the “gateway queue manager”) to use point-to-point messaging or publish-subscribe messaging, one or more WebSphere MQ links are created and configured as part of the task. However the wizard does not ask you to set all possible properties of a WebSphere MQ link, and some of the link properties are not set or are set to default values. You can fine tune these properties by modifying the link.

When you create a new WebSphere MQ link, you can also choose not to create sender or receiver channels. These channels can be added later by modifying the link.

When you use the foreign bus connection wizard to create a WebSphere MQ link for point-to-point messaging, you can later modify the link for publish-subscribe messaging by adding a publish/subscribe broker profile to the link.

1. In the navigation pane, click, **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name***.
2. Modify the properties of the link.

   For information about all the properties that you can modify, see WebSphere MQ link [Settings].

   When you create a new WebSphere MQ link, the following properties are set to default values:

   Description
   An optional description for the WebSphere MQ link, for administrative purposes.

   Adoptable
   Whether or not a running instance of a WebSphere MQ link receiver channel (associated with this MQ link) should be adopted or not. In the event of a communications failure, it is possible for a running instance of a WebSphere MQ link receiver channel to be left waiting for messages. When communication is re-established, and the partner WebSphere MQ sender channel next attempts to establish a session with the WebSphere MQ link receiver channel, the request will fail as there is already a running instance of the WebSphere MQ link receiver channel that believes it is in session with the partner WebSphere MQ sender channel. You can overcome this problem by selecting this option, which causes the already running instance of the WebSphere MQ link receiver channel to be stopped and a new instance to be started. By default, this option is not selected.

   Exception destination
   The destination to which a message is forwarded by the system when it cannot be delivered to this destination. By default, exceptions are sent to the “System” destination. However, it can aid problem-solving if you separate these exceptions out from other system messages by sending them to another destination.

Initial state
Whether the WebSphere MQ link is started or stopped when the hosting messaging engine is first started. Until started, the WebSphere MQ link is unavailable. By default, the value is "Started".

Nonpersistent message speed
The class of service for nonpersistent messages on channels of this WebSphere MQ link. By default, the value is "Fast".

3. Configure the additional properties. You can configure any of the following additional properties of this WebSphere MQ link:
   - Publish/subscribe broker profiles
   - Receiver channel
   - Sender channel
   - Sender channel transmitters
4. Configure the related items. You can configure any of the following related items of this WebSphere MQ link:
   - Foreign bus connection
   - Link transmitters
5. Save your changes to the master configuration.
6. If you have enabled dynamic configuration updates, the changes take effect immediately (or on channel restart if you also modified WebSphere MQ link sender or receiver channels), otherwise restart the application server.

Adding or modifying a publish/subscribe broker on the WebSphere MQ link:

Defining a broker profile, part of the WebSphere MQ link, that forms a publish/subscribe bridge with a WebSphere MQ network.

Before you begin

You need to know the name of the bus, messaging engine name, and the name of the WebSphere MQ link on which you intend to create or modify the broker profile. You also need to know the queue manager name for the message broker in the WebSphere MQ network where the input queues for the required publication message flows are located.

About this task

A publish/subscribe broker profile, and associated topic mappings, allows publication and subscription with a broker in the WebSphere MQ network.

When you use the foreign bus connection wizard to connect a bus and a WebSphere MQ queue manager to use publish-subscribe messaging, you can define a publish/subscribe broker profile and associated topic mappings. Alternatively, you can use the foreign bus connection wizard to connect a bus and a WebSphere MQ queue manager to use point-to-point messaging, then later modify the WebSphere MQ link for publish-subscribe messaging by adding a publish/subscribe broker profile and associated topic mappings to the link.

After you have created the broker profile you must ensure that the service integration bus has sufficient authority on the message broker instance to send subscription requests.

1. In the navigation pane, click one of the following paths:
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles**

- **Service integration → Buses → *bus_name* → [Topology] Messaging engines → *engine_name* → [Additional properties] WebSphere MQ links → *link_name* → [Additional Properties] Publish/subscribe broker profiles**

2. In the content pane, either click **New** to add a new broker profile, or click the name of a existing broker profile that you want to modify.
3. Add or modify the properties of the broker profile.

   For information about the broker profile properties (name, description, broker queue manager name), see Publish/subscribe broker profiles [Settings].

   For an existing broker profile you can only modify the description. Note that, for an existing broker profile created through using the foreign bus connection wizard, the broker profile name was generated automatically by adding "_broker_profile" to the end of the broker queue manager name.
4. Optional: Under Additional properties, configure the topic mappings for this broker profile. For more information, see "Adding or modifying topic mappings on the WebSphere MQ link publish/subscribe broker" on page 1625.
5. Click **OK**.
6. Save your changes to the master configuration.

What to do next

After you have created the broker profile you must ensure that the service integration bus has sufficient authority on the message broker instance to send subscription requests. You can do this either by modifying the message broker configuration on the WebSphere MQ network, or by modifying the service integration bus configuration. See "Defining permissions for a WebSphere MQ link publish/subscribe broker to work with WebSphere MQ."

Defining permissions for a WebSphere MQ link publish/subscribe broker to work with WebSphere MQ:

This topic gives a choice of ways to ensure that a service integration bus has authority with a message broker (in a WebSphere MQ network) to send subscription requests.

Before you begin

Before you start this task, you must have created a broker profile, part of a publish/subscribe bridge on a WebSphere MQ link.

About this task

When you have created a broker profile you must ensure that the service integration bus has sufficient authority on the message broker instance to send subscription requests. You can do this either by:

- Modifying the message broker configuration in the WebSphere MQ network if you are a WebSphere MQ administrator. Or
- Modifying the service integration bus configuration if you are a WebSphere Application Server administrator.

To ensure authority, do one of the following steps:

1. Modify the message broker configuration by granting WebSphere MQ permissions for the user ″SIBServer″, which is the user name under which the publish/subscribe bridge control messages will be published.
2. Modify the service integration bus by setting the OutboundUserID of the foreign bus that represents the WebSphere MQ network (which you defined as part of the WebSphere MQ link configuration). The OutboundUserID should be set to a user ID that already has the relevant WebSphere MQ permissions

on the message broker install image. By doing this you can be sure the publish/subscribe bridge control messages will arrive at the message broker in the WebSphere MQ network with the appropriate user ID set in them.

What to do next

You are now ready to define topic mappings on the publish/subscribe broker profile.

Adding or modifying topic mappings on the WebSphere MQ link publish/subscribe broker:

Define topic mappings on a publish/subscribe broker profile, part of the publish/subscribe bridge on a WebSphere MQ link. A topic mapping is a mapping between a topic on a service integration bus and a stream queue and subscription point provided by a WebSphere MQ broker.

Before you begin

You need to know the bus name, messaging engine name, WebSphere MQ link name and the name of the broker profile on which you intend to define the topic mappings.

About this task

Topic mappings are associated with a WebSphere MQ link publish/subscribe broker profile. Together, the profile and topic mappings enable publication and subscription with a broker in a WebSphere MQ network.

Note: Publication messages forwarded to a message broker in the WebSphere MQ network are republished on the same topic as they were originally published to in the service integration bus topic space, and vice versa.

To define topic mappings, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles** → ***profile_name*** → **[Additional Properties] Topic mappings**
2. In the content pane, either click **New** to add a new topic mapping, or click the name of a existing topic mapping that you want to modify. The Topic Mapping [Settings] form is displayed.
3. Type the topic name.

   The name of the topic on the service integration bus. The name must be the same as the topic name on the message broker in a WebSphere MQ network.

   The topic name can contain wild cards that are in the service integration bus syntax. For more information, see Wild cards in topic mapping.
4. Select the name of the topic space that contains the topic. If you don't select a name the default is used.
5. Select the direction of publication flow. The direction of publication flow can be:

   Bi-directional
   Messages flow in both directions between the bus and WebSphere MQ.

   To WebSphere MQ
   Messages flow only from the bus to WebSphere MQ. That is, from WebSphere Application Server to a message broker in the WebSphere MQ network.

   From WebSphere MQ
   Messages flow only to the bus from WebSphere MQ. That is, from a message broker in the WebSphere MQ network to WebSphere Application Server.
6. Select the broker stream queue you want to use. If the queue you require is not on the list, click **other, please specify** then enter the name of the broker stream queue.

The broker stream queue in this instance is a queue on the WebSphere MQ queue manager to which the message broker is connected. This queue is being used as the input node for a message flow containing a publication node. Messages sent to this queue are processed by the message broker, then published to applications that have subscribed on the topic specified in the message.

Stream names are case sensitive.

After you type a new name, then save your changes, the name becomes available for selection in the drop-down list.

7. Select the WebSphere MQ message broker subscription point from which the service integration bus receives messages. If the subscription point you require is not on the list, click **other, please specify** then enter the name of the subscription point.

   The default subscription point is used if no value is specified.

   After you type a new name, then save your changes, the name becomes available for selection in the drop-down list.
8. Click **OK**.
9. Save your changes to the master configuration.
10. Optional: If you have enabled dynamic configuration updates, the changes take effect immediately. Otherwise, restart the application server.

Adding or modifying a WebSphere MQ link receiver channel:

How you can define the properties of the receiver channel on a WebSphere MQ link. This channel receives messages from the WebSphere MQ queue manager or queue-sharing group (known as the "gateway queue manager"). The receiver channel communicates with a WebSphere MQ sender channel on the gateway queue manager, and converts MQ format messages to service integration bus messages.

Before you begin

You need to know the name of the bus, and the messaging engine on the bus that contains the WebSphere MQ link with the receiver channel you intend to add or modify.

About this task

When you use the foreign bus connection wizard to connect a bus and a gateway queue manager to use point-to-point messaging or publish-subscribe messaging, the wizard does not ask you to set all possible properties of a WebSphere MQ link. If you did not choose to **Enable Service integration bus to WebSphere MQ message flow** in the foreign bus connection wizard, you have not yet defined a Websphere MQ receiver channel. If you did choose this option in the wizard, you have defined the receiver channel name, host name and port number, and the wizard has set the other properties to default values.

To add or modify a WebSphere MQ link receiver channel, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Receiver channel**.
2. In the content pane, either click **New** to add a new receiver channel, or click the name of a existing receiver channel that you want to modify.
3. Add or modify the properties of the channel.

   For information about all the properties that you can modify, see WebSphere MQ link receiver channel [Settings].

   If you used the foreign bus connection wizard to create the channel, the wizard has used default values for the following properties:

Inbound nonpersistent message reliability
The acceptable reliability of message delivery for nonpersistent message flows from WebSphere MQ through this WebSphere MQ link, from Best effort to Reliable, in order of increasing reliability. By default, the value is “Reliable”.

Inbound persistent message reliability
The acceptable reliability of message delivery for inbound persistent message flows from WebSphere MQ through this WebSphere MQ link, from Reliable to Assured, in order of increasing reliability. By default, the value is “Assured”.

Prefer queue points local to this link’s messaging engine
When this check box is selected, the link prefers to send inbound messages to available queue points of target queue destinations that are located on the same messaging engine as the link. By default the check box is selected, which corresponds to the behavior in WebSphere Application Server Version 6.x and can make it easier to handle links in a mixed-version cell. If you clear the check box, preference is not given to local queue points and inbound messages are workload balanced across all available queue points of target queue destinations. This option (not to give preference to local queue points) is available only on links running on WebSphere Application Server Version 7.0.

Initial state
Whether the receiver channel is started or stopped when the associated WebSphere MQ link is first started. Until started, the channel is unavailable. By default, the value is “Started”.

4. Click **OK**.
5. Save your changes to the master configuration.
6. Restart the application server.

Adding or modifying a WebSphere MQ link sender channel:

How you can define the properties of the sender channel on a WebSphere MQ link. This channel sends messages to the WebSphere MQ queue manager or queue-sharing group (known as the “gateway queue manager”). The sender channel communicates with a WebSphere MQ receiver channel on the gateway queue manager, and converts service integration bus messages to MQ format messages.

Before you begin

You need to know the name of the bus, and the messaging engine on the bus that contains the WebSphere MQ link with the sender channel you intend to add or modify.

About this task

When you use the foreign bus connection wizard to connect a bus and a gateway queue manager to use point-to-point messaging or publish-subscribe messaging, the wizard does not ask you to set all possible properties of a WebSphere MQ link. If you did not choose to **Enable WebSphere MQ to Service integration bus message flow** in the foreign bus connection wizard, you have not yet defined a WebSphere MQ sender channel. If you did choose this option in the wizard, you have defined the sender channel name, host name, communication port and transport chain, and the wizard has set the other properties to default values.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Sender channel**.
3. In the content pane, either click **New** to add a new sender channel, or click the name of a existing sender channel that you want to modify.
4. Add or modify the properties of the channel.

For information about all the properties that you can modify, see WebSphere MQ link sender channel [Settings].

If you used the foreign bus connection wizard to create the channel, the wizard has used default values for the following properties:

Disconnect interval
The time in seconds for which the sender channel waits for new messages to arrive on the transmission queue after sending a batch of messages. The channel disconnects after this interval, and must be restarted manually or by triggering. By default, the value is 900 seconds.

Short retry count
The maximum number of times that the sender channel tries to restart after a communication or partner failure. If the count of remaining retries reaches zero, and the channel has not restarted, then the long retry mechanism is invoked. By default, the value is 10.

Short retry interval
The number of seconds between attempts by the sender channel to restart after a communication or partner failure. By default, the value is 60 seconds.

Long retry count
The maximum number of times that the sender channel tries to restart after the short retry mechanism did not recover from a communication or partner failure. If the count of remaining retries reaches zero, and the channel has not restarted, then an error is logged and the channel is stopped. By default, the value is 999999999.

Long retry interval
The number of seconds between attempts by the sender channel to restart after the short retry mechanism did not recover from a communication or partner failure. By default, the value is 1200 seconds.

Initial state
Whether the sender channel is started or stopped when the associated WebSphere MQ link is first started. Until started, the channel is unavailable. By default, the value is “Started”.

5. Click **OK**.
6. Save your changes to the master configuration.
7. Restart the application server.

Modifying security for a WebSphere MQ link:

Securing access between a service integration bus and a WebSphere MQ Queue Manager.

About this task

When you create a new WebSphere MQ link, you can use the foreign bus connection wizard to enable security:

- If the WebSphere MQ queue manager requires a secure connection, you can set the WebSphere MQ receiver channel to accept only connections that have secure sockets layer (SSL) based encryption.
- If the local bus is secure, you can set the service integration bus inbound user ID to replace the user ID in messages from the WebSphere MQ queue manager, so that these messages are authorized to access their destinations.

Use this task to secure the local and foreign bus that are part of a WebSphere MQ links configuration, and to secure an existing WebSphere MQ link that was not secured when it was first created.

For more general information about service integration bus security, see “Security” on page 1128.

1. Enable security on the service integration bus and the foreign bus representing the WebSphere MQ network. See “Disabling bus security” on page 1273.

2. Secure the link between the buses - see Securing connections to a WebSphere MQ network.
3. Grant access to the local bus for users who will be sending messages to the foreign bus - see “Disabling bus security” on page 1273.
4. Grant access to the foreign bus for users who will be sending messages to it - see “Administering foreign bus roles” on page 1292.
5. Optional: Give users access to foreign or alias destinations that will forward messages to a foreign bus - see “Administering destination roles” on page 1241.

Viewing the status of a WebSphere MQ link and its sender and receiver channels:

This topic describes how you can view the status of a WebSphere MQ link and its components, such as the sender and receiver channels.

Before you begin

You may want to view the status of a WebSphere MQ link or its components because:

- You intend to stop the WebSphere MQ link and you need to see if there are any messages currently held on it.
- You are about to delete the WebSphere MQ link and you need to verify there are no messages on it.
- Messages have not arrived at their expected destination and you want to check if they are being held, pending transmission over the WebSphere MQ link.

Note: If you attempt to start a WebSphere MQ sender channel that is already in RUNNING state, the WebSphere Application Server administrative console might occasionally incorrectly report the channel status as INACTIVE. If this happens, you can ignore the error because the channel is still running. To return the channel status to RUNNING, stop and then restart the channel.

About this task

To view the status of a WebSphere MQ link, use the administrative console to complete the following steps:

1. In the navigation pane, click one of the following paths:
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links**
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links**
2. Click the WebSphere MQ link you want to view.
3. Click the **Runtime** tab. The content pane displays the status of the WebSphere MQ link.
4. Click the **Configuration** tab.
5. To view the status of the sender channel:
   a. Click **Sender channel**.
   b. Click the channel you want to view.
   c. Click the **Runtime** tab. The content pane displays status information for the sender channel. Some of the more important fields to check are:
      - The status of the channel.
      - Whether or not the message on the channel is in doubt. If the message is in doubt, it has not been acknowledged by WebSphere MQ.
      - The number of messages in the current batch.
      - The number of batches that have been sent.
      - The time and date at which the channel was last started.
      - The time and date at which the last message was sent.

d. Click **Saved batch status**. The content pane displays the saved status of message batches that have been saved for transmission to WebSphere MQ.
e. If a batch is in doubt, you can either commit or rollback the batch.

6. To view the status of the receiver channel:
   a. Return to the WebSphere MQ link page.
   b. Click **Receiver channel**.
   c. Click the **Runtime** tab. The content pane displays the status of the receiver channel.
   d. Click **Receiver channel connections**. The content pane displays the connections existing on the receiver channel, and their current status. You can stop connections if required, by selecting the check box next to the connection and clicking **Stop**.
   e. Return to the receiver channel page.
   f. Click **Saved batch status**. The content pane displays the saved status of message batches that have been received from WebSphere MQ.

What to do next

You can also view the status of subscriptions for a broker profile on the WebSphere MQ link: "Viewing the status of subscriptions for a WebSphere MQ link publish/subscribe broker profile."

Viewing the status of subscriptions for a WebSphere MQ link publish/subscribe broker profile:

This topic describes how you can view the status of subscriptions for a broker profile on a WebSphere MQ link.

1. In the navigation pane, click one of the following paths:
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles**
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles**
2. Click the broker profile containing the subscriptions you want to view.
3. Click the **Runtime** tab. The content pane displays the current number of subscriptions for the broker profile.
4. Click **Subscriptions**. The content pane displays the status of the subscriptions for the broker profile. Position the mouse over a status icon to view hover help for that icon. You can remove the runtime subscriptions if required, by clicking **Unsubscribe**.

States of the WebSphere MQ link and its channels:

This topic describes the various states of the WebSphere MQ link and its sender and receiver channels, on a service integration bus.

State of WebSphere MQ link	WebSphere MQ link sender channel	WebSphere MQ link receiver channel
INACTIVE	Same as STOPPED for the WebSphere MQ link sender. If the administrator requests that the channel go into target state STOPPED it will transition into the STOPPED state. Similarly, an administrator starting the channel will cause it to transition into STANDBY state.	No network connection exists between the application server and the queue manager. If the attempts of a WebSphere MQ sender channel to establish a connection should succeed, it will become possible for messages to flow from the queue manager to the messaging engine. In this case the receiver channel will transition into RUNNING state.
STARTING	A transitional state. The channel should successfully pass through this into BINDING state with no intervention.	A transitional state. The channel should successfully pass through this into BINDING state with no intervention.
BINDING	A transitional state. The channel should successfully pass through this into RUNNING state with no intervention. The channel may transition into STOPPING state if a problem occurs.	A transitional state. The channel should successfully pass through this into RUNNING state with no intervention. The channel may transition into STOPPING state if a problem occurs.
INITIALIZING	A transitional state. The channel should successfully pass through this into STARTING state with no intervention.	A transitional state. The channel should successfully pass through this into STARTING state with no intervention.
RETRYING	A network connection to the queue manager has been lost. Whilst in this state the channel is attempting to reestablish the connection. If the retry intervals are exhausted without successfully establishing the connection then the channel transitions into STOPPED state. If a connection is successfully reestablished, the channel enters INITIALIZING state.	Not applicable to receiver channel.
STANDBY	When in this state, the sender channel is not network-connected to its WebSphere MQ counterpart receiver channel. It is waiting for a message to send before attempting to establish a connection. When a message arrives for transmission, the channel will transition into STARTING state and start the process of attempting to establish a network connection. The administrator may command the channel to transition into either INACTIVE or STOPPED state from this state.	Not applicable to receiver channel.

State of WebSphere MQ link	WebSphere MQ link sender channel	WebSphere MQ link receiver channel
RUNNING	In this state a network connection has been established between application server and queue manager. Messages destined for the queue manager will be transmitted. Either attempting to stop the channel by using the administrative console, or the loss of the network connection will cause transition into the STOPPING state.	Network connection established between application server and queue manager. Messages destined for the messaging engine will be received. Attempting to stop the channel, or the loss of the network connection will cause a transition into STOPPING state.
STOPPING	A transitional state. The channel should transition into either RETRYING state or STOPPED state without intervention. If the channel has been placed in STOPPING state by the administrative request to become INACTIVE then it will transition into the STANDBY state. If the channel has been placed in this state by administrator request to stop, it will transition into the STOPPED state. If the channel has been placed in this state by a broken network connection it will transition into the RETRYING state, assuming that it has non-zero retry intervals or is otherwise STOPPED.	A transitional state. The channel will transition from this state into STOPPED without intervention.
STOPPED	No network connection exists between the application server and the queue manager. Messages destined for the queue manager will not be transmitted. To transition from this state requires the administrator to start the channel, this will place it in STANDBY state.	No network connection exists between the application server and the queue manager. Any attempt by a sender channel in the WebSphere MQ network to establish a connection will be rejected. Messages destined for the messaging engine will not be received. Administrator action is required to move the channel out of this state. Starting the channel will move it into INACTIVE state.

For information about the states of the channels in a WebSphere MQ network refer to the Intercommunications book available at the WebSphere MQ publications site

Starting a WebSphere MQ link:

Change the state of a WebSphere MQ link from stopped to started.

Before you begin

You might want to tell the WebSphere MQ administrator that you are about to start this WebSphere MQ link.

About this task

To start a WebSphere MQ link, use the administrative console to complete the following steps.

1. In the navigation pane, click one of the following paths:

- **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links**
- **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links**

2. In the content pane, select the check box next to the WebSphere MQ link you want to start.
3. Click **Start**.

Results

If the WebSphere MQ link starts successfully, the status icon changes to indicate that the WebSphere MQ link is running.

Stopping a WebSphere MQ link:

This topic describes how you change the state of a WebSphere MQ link from started or running to stopped.

Before you begin

You might want to tell the WebSphere MQ administrator that you are about to stop this WebSphere MQ link.

About this task

When you stop a WebSphere MQ link, all communication with the target WebSphere MQ network is stopped for both point-to-point and publication and subscription. Messages waiting for transmission are held on the service integration bus, and the MQ sender channel cannot start. If there is a publish/subscribe bridge on the WebSphere MQ link, its operations are stopped.

You can also stop either the sender or receiver channel on the WebSphere MQ link, while leaving the link itself running. See “Stopping the sender channel on a WebSphere MQ link” on page 1635, and “Stopping the receiver channel on a WebSphere MQ link” on page 1634.

To stop a WebSphere MQ link, use the administrative console to complete the following steps.

1. In the navigation pane, click one of the following paths:
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Foreign bus connections** → ***foreign_bus_name*** → **[Related Items] WebSphere MQ links**
   - **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links**
2. In the content pane, select the check box next to the WebSphere MQ link you want to stop.
3. In the **Stop mode** list, select Quiesce or Force.
4. In the **Target state** list select **Inactive** or **Stopped**.
5. Click **Stop**.

Results

The resultant state of the WebSphere MQ link, its sender and receiver channels, and the WebSphere MQ sender channel which is connected to the WebSphere MQ link receiver channel, depends on the options you chose:

- Stopping the WebSphere MQ link to target state INACTIVE will cause the sender channel to go into state STANDBY and the receiver channel to go into state INACTIVE. The overall WebSphere MQ link status will be RUNNING.

- Stopping the WebSphere MQ link to target state STOP will cause the sender channel to go into state STOPPED and the receiver channel to go into state STOPPED. The overall WebSphere MQ link status will be STOPPED.

These are the final states that the sender, receiver and WebSphere MQ link transition into. If you specify a mode of QUIESCE for either of the two final states, the channels and link might not transition into their final states immediately. Instead they temporarily transition through other states required to reach their final state.

For more details about stopped states of the WebSphere MQ link, see “States of the WebSphere MQ link and its channels” on page 1630.

What to do next

You can restart the WebSphere MQ link by selecting the link again and clicking **Start**. When the WebSphere MQ link is stopped and not in doubt, you can delete the WebSphere MQ link and any associated publish/subscribe broker profiles and topic mappings as described in “Removing a foreign bus connection from a bus” on page 1171.

Stopping the receiver channel on a WebSphere MQ link:

This topic describes how to stop the receiver channel on a WebSphere MQ link while leaving the link itself running.

Before you begin

Note: If you stop the receiver channel on a WebSphere MQ link, communication with the target WebSphere MQ network on that channel will cease for both point-to-point messaging and publishing and subscribing. Messages will be held at their transmission locations.

You may wish to warn the administrator of the WebSphere MQ network that you are about to stop the channel.

About this task

If you stop a receiver channel, messages sent to the WebSphere MQ link engine are not received.

If a WebSphere MQ sender channel is started while an MQ link receiver channel is stopped, the request fails with an error indicating that the receiver channel is not available.

For more information on stopped states of the WebSphere MQ link and its channels, see “States of the WebSphere MQ link and its channels” on page 1630.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Receiver channel**.
3. Select the check box next to the channel you want to stop.
4. In the **Stop mode** list, select Quiesce or Force.
5. In the **Target state** list, select Inactive or Stopped.
6. Click **Stop**.

Results

Stopping a receiver channel stops all the receiver channel connections for that receiver. The resultant state of the receiver channel, and the sender channel in the WebSphere MQ network with which it is

communicating, depends on the options you choose:

		Stop mode	
		Quiesce	Force
Target state	Inactive	The receiver channel moves to the stopping state and the data flow to the WebSphere MQ sender channel stops. When the WebSphere MQ sender channel next tries to communicate with the receiver channel the WebSphere MQ sender channel enters a state of retrying. The receiver channel then becomes inactive. The retrying WebSphere MQ sender channel then reestablishes a session with the receiver channel, and both channels become running.	The receiver channel immediately becomes inactive. When the WebSphere MQ sender channel next tries to communicate with the receiver channel, the WebSphere MQ sender channel enters a state of retrying. The retrying WebSphere MQ sender channel then reestablishes a session with the receiver channel, and both channels become running.
	Stopped	The receiver channel moves to the stopping state and the data flow to the WebSphere MQ sender channel stops. When the WebSphere MQ sender channel next tries to communicate with the receiver channel the WebSphere MQ sender channel enters a state of retrying. The receiver channel then becomes stopped, so preventing the WebSphere MQ sender channel from reestablishing a session. The WebSphere MQ sender channel itself then becomes stopped.	The receiver channel immediately becomes stopped. When the WebSphere MQ sender channel next tries to communicate with the receiver channel, the WebSphere MQ sender channel enters a state of retrying, and then becomes stopped itself.

Stopping the sender channel on a WebSphere MQ link:

This topic describes how to stop the sender channel on a WebSphere MQ link while leaving the link itself running.

Before you begin

You might want to tell the WebSphere MQ network administrator that you are about to stop a channel.

About this task

When you stop the sender channel on a WebSphere MQ link, communication with the target WebSphere MQ network on that channel is stopped for both point-to-point messaging and publishing and subscribing. Messages are held at their transmission locations.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Sender channel**.
3. Select the check box next to the channel you want to stop.
4. In the **Stop mode** list, select Quiesce or Force.
5. In the **Target state** list, select Inactive or Stopped.
6. Click **Stop**.

Results

Only the sender channel is affected by this procedure. The resultant state of the sender channel depends on the options you chose:

		Stop mode	
		Quiesce	Force
Target state	Inactive	The sender channel becomes inactive either when it has finished processing its current batch, or when it reaches a heartbeat interval.	The sender channel immediately becomes inactive.
	Stopped	The sender channel becomes stopped either when it has finished processing its current batch, or when it reaches a heartbeat interval.	The sender channel immediately becomes stopped.

For more information on stopped states of the WebSphere MQ link and its channels, see “States of the WebSphere MQ link and its channels” on page 1630.

Deleting a WebSphere MQ link publish/subscribe broker profile:

This describes how you delete a broker profile and all topic mappings on a WebSphere MQ link, which forms a publish/subscribe bridge between WebSphere Application Server and a WebSphere MQ network.

Before you begin

Before you start you need to know the names of the bus, messaging engine, and WebSphere MQ link that has the broker profile that you intend to delete. You should also consider informing the WebSphere MQ administrator that you are about to delete the connection to the message broker in the WebSphere MQ network.

If you also intend to delete the associated WebSphere MQ link, you need not complete this task. Refer instead to “Removing a foreign bus connection from a bus” on page 1171.

About this task

Deleting a broker profile is a three-stage operation to ensure both the application server and the WebSphere MQ network and its message brokers are synchronized after the deletion:

- Remove the subscriptions by unsubscribing the topic mappings on the broker profile.
- When the Runtime view is empty, delete the broker profile.
- If you have enabled dynamic configuration updates, the changes take effect immediately, otherwise restart the application server.

Note: If you remove the subscriptions but do not delete the broker profile, then the subscriptions are recreated when the server is restarted (because they are still present in the static configuration information for the WebSphere MQ link). These subscriptions are unrelated to the original subscriptions so this can lead to some messages in a publication flow being missing for subscribers on the target side of the bridge. For example, any messages published on an unsubscribed topic between the time the unsubscribe took place and the application server was restarted are not republished to the target side of the WebSphere MQ link.

1. Start the administrative console.
2. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles** → ***profile_name***.

3. Remove the subscriptions:
   a. Click the **Runtime** tab.
   b. Click **Subscriptions**.
   c. Click **Unsubscribe** to remove all the subscriptions listed. When an unsubscribe command is sent to the message broker in the WebSphere MQ network, the relevant topic mapping is put into an ″in doubt″ state until the unsubscribe is confirmed when the topic mapping is deleted. After the unsubscribe is confirmed the topic mapping is no longer shown in the runtime view. You might need to refresh the runtime view for all subscriptions to be shown as removed.
4. Delete the broker profile:
   a. Return to the **Publish/subscribe broker profiles** page.
   b. Select the check box next to the broker profile you want to delete.
   c. Click **Delete**.
5. Save your changes to the master configuration.
6. If you have enabled dynamic configuration updates, the changes take effect immediately. Otherwise, restart the application server.

Deleting a topic mapping on a WebSphere MQ link publish/subscribe broker profile:

This describes how you delete a topic mapping on a broker profile on a WebSphere MQ link, which forms part of a publish/subscribe bridge between WebSphere Application Server and WebSphere MQ.

Before you begin

Before you start you need to know the bus name, messaging engine name, WebSphere MQ link name and broker profile name on which you intend to delete the topic mapping.

Note:

- If you intend to delete *all* topic mappings prior to deleting the broker profile or the WebSphere MQ link, you can avoid restarting the application server more than once by following the steps given in “Deleting a WebSphere MQ link publish/subscribe broker profile” on page 1636.
- Deleting a topic mapping is a two-stage operation to ensure both the WebSphere Application Server (base) and the message brokers in the WebSphere MQ network are synchronized once the deletions have taken place:
  1. Delete the topic mapping (see below).
  2. When you have deleted the topic mappings, the changes take effect immediately if you have enabled dynamic configuration updates, otherwise restart the application server.

About this task

To delete a topic mapping, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional properties] WebSphere MQ links** → ***link_name*** → **[Additional Properties] Publish/subscribe broker profiles** → ***profile_name*** → **[Additional Properties] Topic mappings**
2. Select the check box next to the topic mapping you want to delete.
3. Click **Delete**.
4. Save your changes to the master configuration.

What to do next

Any topic mappings that you have deleted are automatically cleaned up by the publish/subscribe bridge when the application server is restarted.

Using a WebSphere MQ server to integrate WebSphere MQ queues into a bus

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. For interoperation with WebSphere Application Server Version 7.0, the version of WebSphere MQ must be WebSphere MQ for z/OS Version 6 or later, or WebSphere MQ (distributed platforms) Version 7 or later.

Before you begin

WebSphere Application Server can interoperate with WebSphere MQ in any of the following ways:

- Using WebSphere MQ as an external JMS provider
- Using a WebSphere MQ link
- Using a WebSphere MQ server

Each type of interoperation is designed for different situations, and provides different advantages. For information about the differences between these approaches, see “Interoperating with a WebSphere MQ network” on page 1612.

Decide on which method you want to use to configure these resources. You can configure WebSphere MQ server resources by using the administrative console as described in this task, or by using the SIBAdminCommands: WebSphere MQ server administrative commands for the AdminTask object.

About this task

To set up and use a WebSphere MQ server, you configure the server properties, add the server to a service integration bus as a bus member, and create a WebSphere MQ queue-type destination. Destinations that are assigned to a WebSphere MQ server bus member can also be mediated.

You add the WebSphere MQ server as a bus member so that messaging engines on the bus can access queues on the target WebSphere MQ system. If your WebSphere MQ server is connected to a queue-sharing group, your bus applications can access shared queues on the target installation.

Note:

- You can configure a WebSphere MQ server to connect to a WebSphere MQ queue manager using either a bindings mode or a client mode connection. To use client mode with WebSphere MQ for z/OS, you need an additional product called the Client Attach Facility.
- You should configure the queues on the WebSphere MQ network as “shareable”. This allows multiple server instances to get messages from the queues.

1. Create a new WebSphere MQ server and configure the server properties.
2. Add the new WebSphere MQ server as a member of a bus so that messaging engines on the bus can access queues on the target WebSphere MQ installation.
3. Create a WebSphere MQ queue type destination for the new bus member and assign it to a WebSphere MQ queue.
4. Optional: Mediate the new destination using the WebSphere MQ queue as the mediation point.

Creating the WebSphere MQ server definition:

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. Creating a WebSphere MQ server involves using the administrative console to define the server’s connection and quality of service properties.

Before you begin

Decide on which method you want to use to configure these resources. You can create a new WebSphere MQ server by using the administrative console as described in this task, or by using the createSIBWMQServer command.

About this task

When you create a new WebSphere MQ server definition, you specify its connection and quality of service properties. When you subsequently add the server as a member of a service integration bus, you can optionally override the server connection settings with the bus connection settings. This means that you can create a WebSphere MQ server that is specific to a bus, yet reusable in a multiple bus topology.

1. Start the administrative console.
2. Navigate to **Servers** → **Server Types** → **WebSphere MQ servers**. The WebSphere MQ servers [Collection] form is displayed.
3. Click **New**. The WebSphere MQ servers wizard starts.
4. Complete the required fields.
   - Name: this is any administrative name that you want to use for this server configuration; ensure that the server name you specify is unique.
   - Server: this is the name of the WebSphere MQ queue or queue-sharing group that this server represents.
   - Connection properties: complete the fields and click **Test connection**.
   - Security: select “Trust user identifiers” if you do not want the user identifiers in messages to be overwritten with the administrative name of the WebSphere MQ server.
   - Resource discovery: Select the check box if you want resources on the WebSphere MQ network to be discovered automatically.
5. Click **OK** to confirm.
6. Save your changes to the master configuration.
7. Restart the application server.

What to do next

You are now ready to add the new WebSphere MQ server as a member of a bus.

Modifying a WebSphere MQ server:

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. This topic describes how to modify a WebSphere MQ server definition.

Before you begin

Decide on which method you want to use to configure these resources. You can modify a WebSphere MQ server by using the administrative console as described in this task, or by using the modifySIBWMQServer command.

About this task

A WebSphere MQ server definition defines the connection to an underlying WebSphere MQ queue manager or queue-sharing group and the associated queues. You can modify some of the properties for this connection.

Note: When you modify a WebSphere MQ server definition, you do not change any configuration values previously inherited from this WebSphere MQ server by existing bus members. For example, suppose you create a WebSphere MQ server with the port number 1234, then add the server to a bus using that server's port number. If you subsequently modify the WebSphere MQ server's port number to 2345, the bus member you previously created is not affected and still has the port number 1234.

1. Start the administrative console.
2. Navigate to **Servers** → **Server Types** → **WebSphere MQ servers** → ***server_name***. The WebSphere MQ server [Settings] form is displayed.
3. Make modifications as necessary.
4. Save your changes to the master configuration.
5. Restart the application server.

Deleting a WebSphere MQ server definition:

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. This topic describes how to delete a WebSphere MQ server definition.

Before you begin

Decide on which method you want to use to configure these resources. You can delete a WebSphere MQ server by using the administrative console as described in this task, or by using the deleteSIBWMQServer command.

Ensure that no application is putting messages to the bus members located on the WebSphere MQ server.

Inform the WebSphere MQ administrator that the WebSphere MQ server is about to be deleted and therefore will no longer interoperate with its WebSphere MQ queue manager or queue-sharing group in the WebSphere MQ network.

About this task

When you delete a WebSphere MQ server definition, the deletion process also modifies the following associated resources:

- It deletes all WebSphere MQ server bus members that were created when the server was added to service integration buses.
- It unmediates all destinations that were assigned to those bus members.
- It removes all queue points that were assigned to those bus members.

Deleting a WebSphere MQ server definition does not affect the associated queue managers, queue-sharing groups, queues or messages on your WebSphere MQ network.

1. Start the administrative console.
2. Navigate to **Servers** → **Server Types** → **WebSphere MQ servers**. The WebSphere MQ servers [Collection] form is displayed.
3. Select the check box next to the WebSphere MQ server you want to delete.
4. Click **Delete**.
5. Save your changes to the master configuration.
6. Restart the application server.

Adding a WebSphere MQ server as a member of a bus:

A WebSphere MQ server provides a direct client connection between a service integration bus and queues on a WebSphere MQ queue manager or (for WebSphere MQ for z/OS) queue-sharing group. A WebSphere MQ server bus member is used as a bus member for assigning queue points and mediation points to WebSphere MQ queues.

Before you begin

Get details of the client connection from your WebSphere MQ administrator.

Ensure that the WebSphere MQ server has been configured, that the bus has been defined and that the server is not already a member of the bus.

Decide on which method you want to use to configure these resources. You can add the WebSphere MQ server as a bus member by using the administrative console as described in this task, or by using the addSIBusMember command.

About this task

When you add a WebSphere MQ server to one or more buses, messaging engines on these buses can access queues on the target WebSphere MQ installation. When you make the server a bus member, you can override the server connection settings with settings that are specific to the new bus member. This can be useful in a multiple bus topology.

1. Start the administrative console.
2. Navigate to the list of bus members for the bus to which you are adding the WebSphere MQ server. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Bus members**.
3. Click **Add**. The “Add a new bus member” wizard is displayed.
4. Select the WebSphere MQ server to add to the bus:
   a. Select the **WebSphere MQ server** radio button.
   b. From the drop-down list, select the server to add.
   c. Click **Next**.
5. Specify the virtual queue manager name.
   When sending messages to WebSphere MQ, the WebSphere MQ gateway queue manager sees the bus as a remote queue manager. The virtual queue manager name is the name that is passed to WebSphere MQ as the name of this remote queue manager. The default value is the name of the bus. If this value is not a valid name for a WebSphere MQ queue manager, or if another WebSphere MQ queue manager already exists that has the same name, then replace the default value with another value that is a valid and unique name for a WebSphere MQ queue manager. To be valid, the name must meet the following criteria:
   - It must contain between 1 and 48 characters.
   - It must conform to the WebSphere MQ queue naming rules (see the WebSphere MQ topic Rules for naming WebSphere MQ objects).
6. Optional: To override the server connection settings, select the **Override WebSphere MQ server connection properties** check box.
   When you select this option, the connection properties for the server are made available so that you can change them to settings that are specific to this bus member. For more information about these connection properties, see WebSphere MQ server bus member [Settings].
7. Optional: If you have changed the server connection settings, you can click **Test connection** to test the connection to the associated WebSphere MQ network.
8. Click **Next**.
9. Click **Finish** to confirm.
10. Save your changes to the master configuration.

What to do next

You are now ready to create a WebSphere MQ queue-type destination for the new bus member.

Creating a queue-type destination and assigning it to a WebSphere MQ queue:

You can use the administrative console to create a queue-type destination and assign it to a WebSphere MQ queue. Select the WebSphere MQ server to host the queue, then specify the WebSphere MQ queue to be hosted.

Before you begin

Get the name of the WebSphere MQ queue from your administrator, and ensure that the following configuration is established:

- The WebSphere MQ server is added as a member of a bus
- The WebSphere MQ queue for the queue point exists
- The WebSphere MQ administrator has set the queue attributes to “shareable”

Note: A shareable queue can be accessed by more than one service integration application.

Decide on which method you want to use to configure these resources. You can create a bus destination by using the administrative console as described in this task, or by using the createSIBDestinations command.

About this task

After you have added a WebSphere MQ server as a bus member, you can create a queue-type destination on the bus member that uses a WebSphere MQ queue as a queue point. This configuration enables service integration applications to send messages to and receive messages from that queue.

1. Start the administrative console.
2. Navigate to the list of destinations for the appropriate bus.

   Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**.

   The Destinations [Collection] form is displayed.
3. Click **New**. The “Create a new destination” panel is displayed.
4. Select **Queue** as the destination type, then click **Next**. The “create a new queue” wizard is displayed.
5. Set the queue attributes. Enter the name that you want WebSphere Application Server to use to refer to the associated WebSphere MQ queue, and (optionally) a description of the queue.
6. Assign the queue to the bus member that is to store and process the messages for the queue.

   Select a WebSphere MQ server bus member from the list of available bus members.
7. Set the WebSphere MQ queue point attributes:
   a. Specify a value in the **WebSphere MQ queue name filter** field, then click **Go**.

      The wizard automatically discovers available WebSphere MQ queues. However, some WebSphere MQ topologies have many thousands of queues defined to a queue manager. Use this filter to limit the number of queues that are listed.

      The default filter value is an asterisk (*). If this value (or no value) is set then all queues, or all queues of a specific type (based on any queue type custom property that is set), are listed. Any other value that you specify must meet the following criteria:

      - It must contain between 1 and 48 characters.
      - It must conform to the WebSphere MQ queue naming rules (see the WebSphere MQ topic Rules for naming WebSphere MQ objects).

You can also use the wildcard character (*) in conjunction with other text. For example, if you enter a value of `PAYROLL*`, then all available queues with names that start with PAYROLL are displayed.

b. Specify a WebSphere MQ queue name.

   Select a queue name from the filtered list. If the list does not include the queue that you want, select the last entry in the list labeled “other, please specify”. A text entry box is displayed next to the list box. Type the queue name into the text entry box.

   If the queue is found on the remote WebSphere MQ system, the properties of the queue as defined within WebSphere MQ are displayed as read-only fields. This should help you to confirm that you have found the queue that you want, and that it is configured as you intend. If the queue is not found, these read-only fields are removed from view.

c. Specify the reliability levels that you require when inbound nonpersistent and inbound persistent WebSphere MQ messages are converted to service integration format messages.

   Applications receive messages direct from the specified WebSphere MQ queue, so in general the reliability level for a message is of no interest to the receiver because the message has already been delivered successfully. However, the message is converted to a service integration format message (and typically to a JMS format service integration message) as it is received, and this option specifies the reliability level for the service integration format message. For information about the available reliability levels, see WebSphere MQ queue points [Settings].

d. Specify whether you want WebSphere MQ to include an MQRFH2 message header when sending messages to the queue.

   The MQRFH2 header stores service integration messaging information that does not have a corresponding WebSphere MQ message header field. When a message is sent to the destination, service integration instructs WebSphere MQ to write the message to the queue. This option specifies whether or not service integration instructs WebSphere MQ to write the message with an MQRFH2 header.

   If the consumer of the message is a JMS application running in WebSphere MQ or service integration, or a WebSphere MQ XMS application, or a WebSphere MQ MQI application that expects an MQRFH2 header, select this option. If the consumer is a WebSphere MQ MQI application that does not expect an MQRFH2 header, do not select this option.

8. Click **Next**.
9. Click **Finish** to confirm queue creation.

Results

You have created a queue-type destination with a WebSphere MQ queue point.

What to do next

You are now ready to (optionally) mediate the new destination using the WebSphere MQ queue as the mediation point.

Mediating a destination using a WebSphere MQ queue as the mediation point:

Mediate a destination by using the administrative console to specify a WebSphere MQ server bus member where the mediation point is to be assigned, and a WebSphere MQ queue to use as the mediation point where messages are stored. To mediate the destination using a service integration mediation, you must also specify a second bus member (not a WebSphere MQ server) to use as the mediation execution point and process the messages.

Before you begin

Decide on which method you want to use to configure these resources. You can mediate a destination by using the administrative console as described in this task, or by using the mediateSIBDestination command.

Before performing this task, ensure that the following resources exist:

- The mediation that you want to apply to the destination
- The WebSphere MQ server bus member where the mediation point is to be assigned
- The WebSphere MQ queue to use as the mediation point
- For a service integration mediation, a second bus member (not a WebSphere MQ server bus member) to use as the mediation execution point where the mediation code runs

Note: When you complete this task, the queue manager on the WebSphere MQ network does not have to be available, but the destination is not usable until the queue manager is available.

About this task

You can mediate a destination with a WebSphere MQ mediation point. This ensures that messages arriving at the designated WebSphere MQ queue are mediated. In this scenario, the mediated messages are delivered to the queue point, or to another destination that is determined by the default forward routing path destination, or by the mediation code.

1. Start the administrative console.
2. Navigate to the list of destinations for the appropriate bus. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Destinations**.
3. Select the check box beside the destination to mediate, then click **Mediate**. The Mediation wizard is displayed.
4. Select the mediation, then click **Next**. From the drop-down list, select the mediation to apply to the destination.
5. Assign the mediation to a bus member:
   a. From the first drop-down list, select the WebSphere MQ server bus member where the mediation point is to be assigned.
   b. From the second drop-down list, select the bus member where the mediation is to run.

   When a mediation is assigned to a WebSphere MQ server bus member, you need a separate bus member that is not a WebSphere MQ server to act as the mediation execution point and process the messages.
6. Set the WebSphere MQ mediation point attributes:
   a. Specify a value in the **WebSphere MQ queue name filter** field, then click **Go**.

      The wizard automatically discovers available WebSphere MQ queues. However, some WebSphere MQ topologies have many thousands of queues defined to a queue manager. Use this filter to limit the number of queues that are listed.

      The default filter value is an asterisk (*). If this value (or no value) is set then all queues, or all queues of a specific type (based on any queue type custom property that is set), are listed. Any other value that you specify must meet the following criteria:

      - It must contain between 1 and 48 characters.
      - It must conform to the WebSphere MQ queue naming rules (see the WebSphere MQ topic Rules for naming WebSphere MQ objects).

      You can also use the wildcard character (*) in conjunction with other text. For example, if you enter a value of `PAYROLL*`, then all available queues with names that start with PAYROLL are displayed.
   b. Specify a WebSphere MQ queue name.

      Select a queue name from the filtered list. If the list does not include the queue that you want, select the last entry in the list labeled “other, please specify”. A text entry box is displayed next to the list box. Type the queue name into the text entry box.

      If the queue is found on the remote WebSphere MQ system, the properties of the queue as defined within WebSphere MQ are displayed as read-only fields. This should help you to confirm

that you have found the queue that you want, and that it is configured as you intend. If the queue is not found, these read-only fields are removed from view.

c. Specify the reliability levels that you require when inbound nonpersistent and inbound persistent WebSphere MQ messages are converted to service integration format messages.

   Mediations receive messages direct from the specified WebSphere MQ queue, so in general the reliability level for a message is of no interest to the mediation because the message has already been delivered successfully. However, the message is converted to a service integration format message (and typically to a JMS format service integration message) as it is received, and this option specifies the reliability level for the service integration format message. For information about the available reliability levels, see WebSphere MQ queue points [Settings].

d. Specify whether you want WebSphere MQ to include an MQRFH2 message header when sending messages to the queue.

   The MQRFH2 header stores service integration messaging information that does not have a corresponding WebSphere MQ message header field. When a message is sent to the destination, service integration instructs WebSphere MQ to write the message to the queue. This option specifies whether or not service integration instructs WebSphere MQ to write the message with an MQRFH2 header.

   If the consumer of the message (in this case, the mediation) is a JMS application running in WebSphere MQ or service integration, or a WebSphere MQ XMS application, or a WebSphere MQ MQI application that expects an MQRFH2 header, select this option. If the mediation is a WebSphere MQ MQI application that does not expect an MQRFH2 header, do not select this option.

7. Click **Next**.
8. Click **Finish** to confirm mediation of the destination.

Results

You have mediated a destination using a WebSphere MQ queue as the mediation point.

Setting the WebSphere MQ encoding format for JMS Map entries

JMS Map messages use the class javax.jms.MapMessage. WebSphere MQ Version 6 provides an encoding format for the body of JMS Map messages that lets you allocate names that are not valid XML tags to Map entries.

Before you begin

This encoding format can interoperate with WebSphere MQ Version 5.3 and above, but not with the earlier WebSphere MQ versions. All WebSphere MQ JMS clients use the WebSphere MQ Version 6 encoding format by default.

About this task

To specify this encoding format, you use the administrative console to set the encoding name/value pair as custom properties of the connection factory. If you specify TRUE the WebSphere MQ Version 5 encoding format is used. If you specify FALSE, the WebSphere MQ Version 6 encoding format is used.

For related information, see the *WebSphere MQ Using Java* book in the WebSphere MQ library.

1. From the administrative console, navigate to **Resources** → **JMS** → **JMS providers** → **WebSphere MQ messaging provider** → **[Additional Properties] Connection factories** → ***factory_name*** → **Custom properties**.
2. Specify the encoding name and value in the Name and Value fields. The encoding name is MAPNAMESTYLE and the value is a Boolean expression, where TRUE is the WebSphere MQ Version 5 compatible format and FALSE is the WebSphere MQ Version 6 format.
3. Restart the application server for the properties to take effect.

Managing WebSphere Application Server Version 5.1 JMS use of messaging resources in later versions of the product

This task enables Java EE applications developed for WebSphere Application Server Version 5.1 to use messaging resources of the default messaging provider in WebSphere Application Server Version 7.0.

Before you begin

Throughout this topic, the abbreviation “Version 5.1” refers to “WebSphere Application Server Version 5.1” and “Version 7.0” refers to “WebSphere Application Server Version 7.0”. For example, “Version 5.1 JMS resources” refers to JMS resources provided by WebSphere Application Server Version 5.1.

This task refers to the *default messaging provider*. For related information, see “Managing messaging with the default messaging provider” on page 1127.

JMS connectivity between the Version 5.1 messaging provider and the default messaging provider in later versions of the product is enabled and managed by a *WebSphere MQ client link*. This does not mean that a WebSphere MQ system is involved. The Version 5.1 messaging provider uses WebSphere MQ client protocols, and is therefore handled as if it were a WebSphere MQ client by the default messaging provider in later versions of the product. The WebSphere MQ client link is provided only for use with JMS applications developed for WebSphere Application Server Version 5.1. Moreover, this JMS connectivity is only intended as an aid to migration from the Version 5.1 messaging provider to the Version 7.0 default messaging provider. For more information about migrating from the Version 5.1 messaging provider, see Migrating from WebSphere Application Server Version 5 embedded messaging.

Applications running in Version 7.0 can use the messaging resources of the Version 5.1 messaging provider without any need for a WebSphere MQ client link.

About this task

To enable JMS applications developed for WebSphere Application Server Version 5.1 to use messaging resources of the default messaging provider, a WebSphere MQ client link is created on the Version 7.0 node. Each WebSphere MQ client link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by Version 5.1 and the protocols used by the default messaging provider in Version 7.0.

The following figure shows a JMS application running on Version 5.1 using JMS resources provided by the default messaging provider on a Version 7.0 node. The JMS queue hosted by Version 5.1 is backed by a service integration bus queue, which is normal for a JMS queue hosted by Version 7.0, but there is no configured link between the Version 5.1 JMS queue and the bus queue. The JMS application communicates with the bus queue through the WebSphere MQ client link and the messaging engine. To send messages to the bus queue or receive messages from the queue, the JMS application opens a connection on the WebSphere MQ client link. This is all invisible to the JMS application, but can be displayed and managed by the administrator.

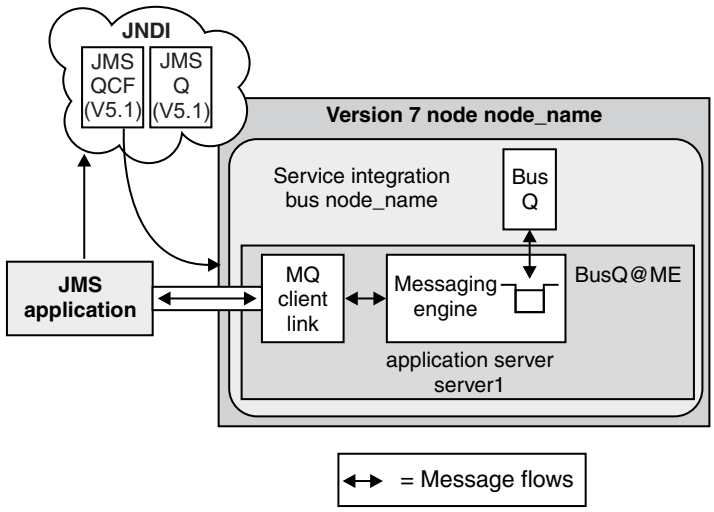


Figure 19. WebSphere Application Server Version 5.1 JMS application scenario

- Configure a Version 7.0 node to support Version 5.1 applications that use JMS resources. If you want a Version 7.0 node to provide JMS destinations for use by applications running on Version 5.1, complete the following steps:
  1. Create an application server. You can use an existing application server on the Version 7.0 node; for example, an application server onto which a Version 5.1 application is to be deployed.
  2. Create a service integration bus. You can use an existing bus.
  3. Add the application server as a bus member. This automatically creates a messaging engine on the application server.
  4. Create a WebSphere MQ client link on the messaging engine. Specify the following property values:

     Name This can be any name that is useful for your administrative purposes. It is not used by the application environment.

     MQ channel name
     This is the name of the channel for the WebSphere MQ client link, used to flow messages between the application that is running on Version 5.1 and the bus. This name must match the receiving channel name configured for Version 5.1:

     `WAS.JMS.SVRCONN`

     This is the default value shown when you first display the WebSphere MQ client link settings panel.

     Queue manager name
     This is the virtual queue manager name that is associated with the messaging engine, and by which the messaging engine is known to applications running on Version 5.1 . Type the queue manager name in the form:

     `WAS_`*`node_name_server_name`*

     Where:

     node_name
     is the name of the Version 7.0 node.

     server_name
     is the name of the application server.

     The correct value is shown by default when you first display the WebSphere MQ client link settings panel.

Default queue manager
Select this check box if you want the WebSphere MQ client link to be used as the default for applications that cannot find a suitable WebSphere MQ client link to use.

If an application running on Version 5.1 specifies that it wants to connect to a non-default queue manager name, you can configure a WebSphere MQ client link with that queue manager name. If a WebSphere MQ client link cannot be found with the required queue manager name, the connection is rejected. Alternatively, you can select this option on another WebSphere MQ client link, which is used instead of rejecting the connection.

5. Define a port called JMSSERVER_QUEUED_ADDRESS on the application server. The port number must be the same used by the SIB_MQ_ENDPOINT_ADDRESS port.

   Specify the following property values:

   Port name
   For **Well-known Port**, select JMSSERVER_QUEUED_ADDRESS

   Host Type the IP address, domain name server (DNS) host name with domain name suffix, or the short DNS host name of the Version 7.0 node.

   Port Type the port number used by the SIB_MQ_ENDPOINT_ADDRESS port. By default, this is 5558.

- Configure a Version 7.0 managed node to support applications running on Version 5.1 that use JMS resources. If you want a Version 7.0 managed node to provide JMS destinations for use by applications running on Version 5.1, complete the following steps:
  1. Create an application server. Specify the name `jmsserver`.
  2. Create a service integration bus. You can use an existing bus.
  3. Add the application server as a bus member. This automatically creates a messaging engine on the application server.
  4. Create a WebSphere MQ client link on the messaging engine. Specify the following property values:

     Name This can be any name that is useful for your administrative purposes. It is not used by the application environment.

     MQ channel name
     This is the name of the channel for the WebSphere MQ client link, used to flow messages between the application running on Version 5.1 and the bus. This name must match the receiving channel name configured for Version 5.1:

     `WAS.JMS.SVRCONN`

     This is the default value shown when you first display the WebSphere MQ client link settings panel.

     Queue manager name
     This is the virtual queue manager name that is associated with the messaging engine, and by which the messaging engine is known to applications running on Version 5.1. Type the queue manager name in the form:

     `WAS_`*`node_name`*`_jmsserver`

     Where:

     node_name
     is the name of the Version 7.0 node.

     The correct value is shown by default when you first display the WebSphere MQ client link settings panel.

     Default queue manager
     Select this check box if you want the WebSphere MQ client link to be used as the default for applications that cannot find a suitable WebSphere MQ client link to use.

If an application developed for WebSphere Application Server Version 5.1 specifies that it wants to connect to a non-default queue manager name, you can configure another WebSphere MQ client link with that queue manager name. If a WebSphere MQ client link cannot be found with the required queue manager name, the connection is rejected. Alternatively, you can select this option on a WebSphere MQ client link, which is used instead of rejecting the connection.

5. Define a port called JMSSERVER_QUEUED_ADDRESS on the application server. The port number must be the same used by the SIB_MQ_ENDPOINT_ADDRESS port.

   Specify the following property values:

   Port name
   For **Well-known Port**, select JMSSERVER_QUEUED_ADDRESS

   Host Type the IP address, domain name server (DNS) host name with domain name suffix, or the short DNS host name of the Version 7.0 node.

   Port Type the port number used by the SIB_MQ_ENDPOINT_ADDRESS port. By default, this is 5558.

- If the application looks up JMS resources in JNDI on the Version 7.0 application server, configure the JMS resources on the Version 7.0 application server as Version 5.1 default messaging JMS resources.
  1. For each JMS queue destination that the application uses, create a Version 5 Default Messaging WebSphere queue destination.
  2. For each JMS queue destination that the application uses, create a bus destination with the same name. Assign the bus destination to a bus member in the same bus as the `jmsserver` bus member.You must also create an alias destination with an identifier `WQ_<destination_name>`, that points to the service integration destination that has been created. The WQ_ prefix is needed because all destination names are prefixed with WQ_. If you are manually migrating the WebSphere JMS provider resources, you also need to create the “WQ_” queues.
  3. Configure JMS connection factories as Version 5.1 default messaging queue connection factories and topic connection factories.
- If the application looks up JMS resources outside the JNDI on the Version 7.0 application server, configure the JMS connection factory to point to the Version 7.0 node.

Results

The application running on Version 5.1 can continue to access the Version 5.1 JMS resources, which are now implemented through the Version 7.0 default messaging provider, as shown in the figure Figure 19 on page 1647. The JMS application communicates with the Version 5.1 JMS resources through the WebSphere MQ client link and the messaging engine. This is invisible to the JMS application. The JMS resources, a JMS queue connection factory, shown as JMS QCF(V5), and a JMS queue, shown as JMS Q(V5), are managed as Version 5.1 default messaging JMS resources. The new bus queue, shown as JMS Q, is managed as a resource of the service integration bus. Messages for JMS Q are stored and processed by the message point for the associated bus destination, a queue shown as Bus Q. The WebSphere MQ client link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by JMS applications developed for WebSphere Application Server Version 5.1 and the protocols used by messaging engines on Version 7.0.

Creating a WebSphere MQ client link

Use this task to create a link that enables a messaging engine to handle message requests from JMS applications developed for WebSphere Application Server Version 5.1.

Before you begin

The WebSphere MQ client link is provided only for use with JMS applications developed for WebSphere Application Server Version 5.1.

About this task

The WebSphere MQ client link enables WebSphere Application Server Version 5.1 Java EE applications to use messaging resources of the WebSphere Application Server Version 7.0 default messaging provider that are backed by destinations on the service integration bus.

Each WebSphere MQ client link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by WebSphere Application Server Version 5.1[10] JMS applications, and the WebSphere Application Server Version 7.0 protocols used by messaging engines.

If you migrate a Version 5.1 node to Version 7.0, a default WebSphere MQ client link, with the name Default.MQClientLink, is created for the application servers on that node. You only need to create another WebSphere MQ client link if you want some Version 5.1 JMS applications to use a different channel to connect to the messaging engine, or want to present a messaging engine as a non-default queue manager name.

If you create a WebSphere MQ client link, you only need to manually specify values for a subset of the available properties. You can choose to specify values for other optional properties in this task, or if wanted at a later time by changing the configuration of the client link.

Note: Creating a WebSphere MQ client link is only part of the task to enable WebSphere Application Server Version 5.1 applications to use messaging resources of the WebSphere Application Server Version 7.0 default messaging provider. For more information about the overall task, see “Managing WebSphere Application Server Version 5.1 JMS use of messaging resources in later versions of the product” on page 1646.

To create a WebSphere MQ client link, use the administrative console to complete the following steps.

1. List the WebSphere MQ client links for the messaging engine that the Version 5.1 JMS applications are to connect to. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links**. This displays any existing WebSphere MQ client links in the content pane.
2. Click **New** in the content pane.
3. Specify the following required properties for the WebSphere MQ client link:

   Name This can be any name that is useful for your administrative purposes. It is not used by the application environment.

   MQ channel name
   This is the name of the channel for the WebSphere MQ client link, used to flow messages between the JMS application developed for WebSphere Application Server Version 5.1 and the bus. This name must match the receiving channel name configured for WebSphere Application Server Version 5.1:

   `WAS.JMS.SVRCONN`

   This is the default value shown when you first display the WebSphere MQ client link settings panel.

   Queue manager name
   This is the virtual queue manager name that is associated with the messaging engine, and by which the messaging engine is known to WebSphere Application Server Version 5.1 applications. Type the queue manager name in the form:

   `WAS_node_name_server_name`

10. To make reading easier in this topic, the abbreviation “Version 5.1” is sometimes used to refer to “WebSphere Application Server Version 5.1” and “Version 7.0” is used to refer to “WebSphere Application Server Version 7.0”. For example, “Version 5.1 JMS resources” refers to JMS resources provided by WebSphere Application Server Version 5.1.

Where:

node_name
is the name of the WebSphere Application Server Version 7.0 node.

server_name
is the name of the application server.

The correct value is shown by default when you first display the WebSphere MQ client link settings panel.

Default queue manager
Select this check box if you want the MQ client link to be used as the default for applications that cannot find a suitable MQ client link to use.

If a JMS application developed for WebSphere Application Server Version 5.1 specifies that it wants to connect to a non-default queue manager name, you can configure a WebSphere MQ client link with that queue manager name. If a WebSphere MQ client link cannot be found with the required queue manager name, the connection is rejected. Alternatively, you can select this option on another WebSphere MQ client link, which is used instead of rejecting the connection.

4. Optional: If you want to specify other optional properties, see the property descriptions in WebSphere MQ client link [Settings].
5. Click **OK**.
6. Save your changes to the master configuration.

What to do next

To enable WebSphere Application Server Version 5.1 applications to use messaging resources of WebSphere Application Server Version 7.0, ensure you complete all the steps in the task “Managing WebSphere Application Server Version 5.1 JMS use of messaging resources in later versions of the product” on page 1646.

Configuring a WebSphere MQ client link

Use this task to browse or change the properties of a client link that enables a messaging engine to handle message requests from JMS applications developed for WebSphere Application Server Version 5.1.

Before you begin

The WebSphere MQ client link is provided only for use with JMS applications developed for WebSphere Application Server Version 5.1.

About this task

The WebSphere MQ client link enables WebSphere Application Server Version 5.1 Java EE applications to use messaging resources of the WebSphere Application Server Version 7.0 default messaging provider that are backed by destinations on the service integration bus.

Each WebSphere MQ client link presents itself as a queue manager and transforms between the WebSphere MQ client protocols used by WebSphere Application Server Version 5.1[11] JMS applications, and the Version 7.0 protocols used by messaging engines.

1. Start the administrative console.

11. To make reading easier in this topic, the abbreviation “Version 5.1” is sometimes used to refer to “WebSphere Application Server Version 5.1” and “Version 7.0” is used to refer to “WebSphere Application Server Version 7.0”. For example, “Version 5.1 JMS resources” refers to JMS resources provided by WebSphere Application Server Version 5.1.

2. Navigate to the settings page for the WebSphere MQ client link. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links** → ***link_name***. This displays the properties of the selected WebSphere MQ client links in the content pane.
3. Optional: If you want to change any of the general properties, see the property descriptions in WebSphere MQ client link [Settings].
4. Optional: If you want to change any of the advanced properties, see the property descriptions in WebSphere MQ client link advanced properties [Settings], then complete the following steps.
5. If you have changed any properties, complete the following steps.
   a. Click **OK**.
   b. Save your changes to the master configuration.

Listing WebSphere MQ client links for a messaging engine

Use this task to display a list of WebSphere MQ links for a messaging engine.

About this task

To list the WebSphere MQ client links for a messaging engine, use the administrative console to complete the following steps.

Display the WebSphere MQ client links collection page. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links**.

Results

This displays a list of WebSphere MQ client links for the messaging engine in the content pane, with the following subset of properties for the link:

Name The name of the WebSphere MQ client link.

Description
An optional description for the WebSphere MQ client link, for administrative purposes.

MQ channel name
The name of the channel for the WebSphere MQ client link, used to flow messages between WebSphere MQ clients and the bus.

Queue manager name
The queue manager name used to represent the bus as a WebSphere MQ queue manager to WebSphere MQ clients.

Default queue manager
Whether or not this is the default queue manager for the WebSphere MQ clients.

Status
The runtime status of the WebSphere MQ client link.

What to do next

You can use this panel to create, delete, start, or stop client links, as described in related tasks.

Starting and stopping WebSphere MQ client links

Use this task to start or stop WebSphere MQ client links, to stop message flows between the default messaging provider and JMS applications developed for WebSphere Application Server Version 5.1, connected through the client links.

About this task

You can start a WebSphere MQ client link that is not running (has a runtime status of Stopped) or stop a client link that is running (has a runtime status of Started). This also starts or stops all client connections on the selected WebSphere MQ client links. Alternatively, you can start or stop individual client connections on the WebSphere MQ client link, as described in “Starting and stopping WebSphere MQ client connections” on page 1654.

When stopping a client link, you can choose whether to force the client link to stop immediately or after any messages currently on its client connections have been processed. You can also choose the final state that you want the client link to be in when it has been stopped.

To start or stop one or more WebSphere MQ client links, use the administrative console to complete the following steps.

1. Display the WebSphere MQ client links collection page. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links**.
2. Select the check box next to the name of each client link that you want to start or stop.
3. Click the appropriate button:

Start	Click this button to start selected items. This enables JMS applications to use the client link to access JMS resources provided by the default messaging provider.
Stop	Click this button to stop a selected WebSphere MQ client link. You must first have selected the link to be stopped. You can choose the mode of stop action and the desired state when the link has been stopped: **Stop mode:** **Force** Stop the link immediately. **Quiesce** Stop the link after any messages currently on its client connections have been processed. **Target state:** **Inactive** Stop the link to an inactive state. If an application tries to use the link, the link is started again. **Stopped** Stop the link to a stopped state. Applications cannot use the client link. The link can only be started again by administrator action.

Results

The status of the client link changes and a message stating that the link has started or stopped is displayed at the top the page.

Listing client connections for a WebSphere MQ client link

Use this task to display a list of client connections for a WebSphere MQ client link. Each client connection is created automatically for a version 5 JMS application using the WebSphere MQ client link.

About this task

To list the client connections for a WebSphere MQ client link, use the administrative console to complete the following steps.

Display the client connections collection page. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links** → ***link_name*** → **[Additional Properties] Client connections**.

Results

This displays a list of client connections for the WebSphere MQ client link in the content pane, with the following subset of properties for the link:

IP address
The TCP/IP IP address of the WebSphere MQ client.

Status
The runtime status of the WebSphere MQ client connection.

What to do next

You can use this panel to start or stop client connections, as described in related tasks.

Starting and stopping WebSphere MQ client connections

Use this task to start or stop client connections on a WebSphere MQ client link, to stop message flows between the default messaging provider and the JMS application developed for WebSphere Application Server Version 5.1 that uses the connections.

About this task

You can start a WebSphere MQ client connection that is not running (has a runtime status of Inactive) or stop a client connection that is running (has a runtime status of Started). Alternatively, you can start or stop all client connections on the WebSphere MQ client link, as described in “Starting and stopping WebSphere MQ client links” on page 1652.

When stopping a client connection, you can choose whether to force the client connection to stop immediately or after any messages currently on its client connections have been processed. Client connections are always stopped to an inactive state.

To start or stop one or more WebSphere MQ client connections, use the administrative console to complete the following steps.

- Display the WebSphere MQ client connections collection page. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links** → ***link_name*** → **[Additional Properties] Client connections**.
- Select the check box next to the name of each client connection that you want to start or stop.
- Click the appropriate button:

Start	Click this button to start selected items.

Stop	Click this button to stop a selected WebSphere MQ client connection. You must first have selected the connection to be stopped. You can choose the mode of stop action, but connections are always stopped to an inactive state: **Stop mode:** **Force** Stop the connection immediately. **Quiesce** Stop the connection after any messages currently on the connection have been processed. **Target state:** **Inactive** Stop the link to an inactive state. If an application tries to use the link, the link is started again.

Results

The status of the client connection changes and a message stating that the connection has started or stopped is displayed at the top the page.

Deleting WebSphere MQ client links

Use this task to delete WebSphere MQ client links. This prevents JMS applications developed for WebSphere Application Server Version 5.1 from using those client links to exchange messages with the default messaging provider.

Before you begin

Before deleting a WebSphere MQ client link, you need to stop version 5 JMS applications using the client link.

- Stop all message-producing JMS applications in the WebSphere Application Server Version 5.1 environment that are using the client link. For example, you can use the administrative console to stop the applications, as described in Starting and stopping applications.
- Allow all message-consuming JMS applications (including those consuming publications as a result of durable subscriptions) to continue until all the JMS queues are drained, then stop those applications.

About this task

To start or stop one or more WebSphere MQ client links, use the administrative console to complete the following steps.

1. Display the WebSphere MQ client links collection page. Click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] WebSphere MQ client links**.
2. Select the check box next to the name of each client link that you want to delete.
3. Click **Delete**
4. Click **OK**.
5. Save any changes to the master configuration.

Configuring the messaging engine selection process for JMS applications

Configure the JMS connection factory for your application, in order to tune the process through which messaging engine connections are selected for your application.

About this task

To use JMS destinations of the default messaging provider, a client application connects to a messaging engine on the service integration bus to which the destinations are assigned. For example, a JMS queue is assigned to a queue destination on a service integration bus.

By default, the environment automatically connects applications to an available messaging engine on the bus. However you can specify extra configuration details to influence the connection process; for example to identify special bootstrap servers, or to limit connection to a subgroup of available messaging engines, or to improve availability or performance, or to ensure sequential processing of messages received.

For a JMS application, you apply the extra configuration to the associated JMS connection factory. For a message-driven bean (MDB) application, you apply the equivalent extra configuration to the associated activation specification.

For the default configuration, you only need to specify the one required connection property **Bus name**, which sets the name of the bus to which the application wants to connect. To further restrict the range of messaging engines to which your applications can connect, you can also configure the other connection properties:

- Target
- Target type
- Target significance
- Target inbound transport chain
- Connection proximity

For detailed information about these connection properties, and an overview of the process through which the default messaging provider chooses the messaging engine for your application, see How JMS applications connect to a messaging engine on a bus.

The steps for this task are based on an application that uses a unified JMS connection factory. You can use the same task to configure a JMS queue connection factory or JMS topic connection factory, but during the task you need to select the appropriate type of connection factory instead of **JMS connection factory**. To configure any of these properties of a connection factory, complete the following steps:

- If the client application uses a JMS connection factory in the client container, use the Client Resource Configuration tool (ACRCT) to configure the Provider endpoint property:
  1. Start the tool and open the EAR file for which you want to configure the JMS connection factory. The EAR file contents are displayed in a tree view.
  2. From the tree, select the JAR file in which you want to configure the JMS connection factory.
  3. Expand the JAR file to view its contents.
  4. Expand **Messaging Providers > Default Provider > Connection Factories**.
  5. Optional: Display the general properties of the connection factory:
     - If you want to use an existing JMS connection factory, click the name of the connection factory.
     - If you want to create a new JMS connection factory, click **New**.

       For more information about configuring a JMS connection factory in the JMS provider configuration for your application client, see Configuring new JMS providers with the Application Client Resource Configuration Tool.

6. On the General tab, configure the connection properties.
7. Click **OK**.
8. To save your changes, click **File > Save**.

- If the client application uses a JMS connection factory on the server, use the WebSphere Application Server administrative console to configure the connection properties:
  1. Display the default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
  2. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
  3. In the content pane click **Default messaging provider**. This displays a table of properties for the default messaging provider, including links to the types of JMS resources that it provides.
  4. In the content pane, under Additional Properties, click **Connection factories** This displays any existing connection factories in the content pane.
  5. Optional: Display the general properties of the connection factory:
     – If you want to use an existing JMS connection factory, click the name of the connection factory.
     – If you want to create a new JMS connection factory, click **New**.

     For more information about configuring a JMS connection factory, see Configuring a unified JMS connection factory for the default messaging provider.
  6. Configure the connection properties.
  7. Click **OK**.
  8. Save your changes to the master configuration.

Managing messages and subscriptions for JMS destinations

Use the following tasks to manage the messages and subscriptions that exist for JMS destinations of the default messaging provide.

About this task

- You can manage the messages on a JMS queue by acting on the queue point for the bus destination to which JMS queue has been assigned. The **Bus name** property of the JMS connection factory identifies the service integration bus. The **Queue name** property of the JMS queue identifies the name of the bus destination.
- You can manage the durable subscriptions on a JMS topic by acting on a publication point for the topic space to which JMS topic has been assigned. The **Bus name** property of the JMS connection factory identifies the service integration bus. The **Topic space** property of the JMS topic identifies the name of the topic space.

Other tasks that affect the runtime status of the default messaging provider are given in related tasks.

To change the configuration of JMS resources for the default messaging provider, see “Configuring resources for the default messaging provider” on page 1584.
- “Managing messages on message points” on page 1194
- “Administering durable subscriptions” on page 1251

Managing messages on message points

Use these tasks to list and act on runtime messages that exist on message points in a service integration bus.

About this task

You can list the message points for bus destinations and messaging engines, and list the messages on a selected message point. You can use the list of messages as part of a troubleshooting task to find messages that need to be deleted.

Listing messages on a message point:

Use this task to list the messages that exist on a message point for a selected bus destination or messaging engine.

About this task

To display a list of messages on a message point, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses**.
2. In the content pane, click the name of the service integration bus.
3. Optional: To list the message points for a bus destination, complete the following steps:
   a. In the content pane, under **Destination resources**, click **Destinations**.
   b. Click the destination name.
4. Optional: To list the message points for a messaging engine, complete the following steps:
   a. In the content pane, under **Topology**, click **Messaging engines**
   b. Click the messaging engine name.
5. Under Additional Properties, click **Message points** This displays a list of message points in the content pane.
6. Click the message point name. This displays the properties of the destination localization in the content pane.
7. Click the Runtime tab.
8. Under Additional Properties, click **Messages**

Results

A list of messages on the selected message point is displayed in the content pane.

What to do next

You can select one or more messages to act on; for example, to display the message content, delete messages.

Deleting messages on a message point:

Use this task to delete one or messages that exist on a message point for a selected bus destination or messaging engine.

About this task

You should not normally need to delete messages on a message point. This task is intended as part of a troubleshooting procedure.

To delete one or messages on a message point, use the administrative console to complete the following steps:

1. List the messages on the message point.
2. In the content pane, click the check box next to each message you want to delete. Alternatively, you can select all messages in the list by clicking the Select all items button 

3. Click **Delete**

Results

The selected messages are removed from the list.

Using JMS from standalone clients to interoperate with service integration resources

The Thin Client for JMS with WebSphere Application Server allows third party applications to interoperate with default messaging provider messaging engines on WebSphere Application Server.

Using JMS to connect to a WebSphere Application Server default messaging provider messaging engine

The Thin Client for JMS with WebSphere Application Server is an embeddable technology that provides Java Message Service (JMS) V1.1 connections to a WebSphere Application Server default messaging provider messaging engine.

Installing and configuring the Thin Client for JMS with WebSphere Application Server:

To use the Thin Client for JMS with WebSphere Application Server copy the com.ibm.ws.sib.client.thin.jms_7.0.0.jar and any other required files from the application server or application client %WAS_HOME%/runtimes directory.

About this task

The Thin Client for JMS with WebSphere Application Server can be used for default messaging provider messaging engines for WebSphere Application Server Version 6.0.2 or later. The connection to the messaging engine can be either TCP or SSL. HTTP connectivity is not supported.

You can install the client in any location and run it in any supported Java 2 Platform Standard Edition 1.5.0 (also known as 5.0) or above Java Runtime Environment (JRE). The client supports the following JREs:

- IBM JRE 1.5.0 and above
- Sun JRE 1.5.0 and above
- HP-UX JRE 1.5.0 and above
- Lotus Expeditor v6.1 or above with J2SE 5.0 or above Device Runtime Environment. jclDesktop and jclDevice profiles are not supported.

The client does not require any further configuration after installation, apart from adding the jar file or files to the classpaths for your client application. You can choose either to create JMS connection factories programmatically, or use the Java Naming and Directory Interface (JNDI). If required, you can use secure connections by configuring Secure Sockets Layer (SSL) settings.

1. Install the client in the required location. The client is always installed in the /runtimes directory of a WebSphere Application Server installation, and may optionally be installed by the Application Client for WebSphere Application Server, which is a separate WebSphere Application Server deliverable. The client is shipped as three files:
   - com.ibm.ws.sib.client.thin.jms_7.0.0.jar - the regular JMS Client.
   - com.ibm.ws.sib.client_ExpeditorDRE_7.0.0.jar - the JMS Client packaged for Lotus Expeditor.
   - sibc.nls.zip - language-specific resource bundles. You can extract any combination of these files. The client jar already includes US English, so you only need the additional language files from sibc.nls.zip if you require languages other than non-US English.
2. Include the appropriate jar file or files in the classpaths for your client application:
   a. To compile JMS code, include the client jar file in the CLASSPATH setting for the javac command.

b. To run JMS code, include the client jar file and any required optional language files extracted from sibx.nls.zip in the CLASSPATH setting for the java command.

3. Configure the required JMS resources as described in “Using JMS resources with the Thin Client for JMS with WebSphere Application Server” on page 1661.
4. If you require secure connections, configure SSL as described in “Securing client and resource adapter connections” on page 1662.

Migration to Thin Client for JMS with WebSphere Application Server:

There are a number of differences to consider when migrating to the Thin Client for JMS with WebSphere Application Server from an earlier version of the client.

Table 16. Migration from WebSphere Application Server Version 6.0.2 to WebSphere Application Server Version 7.0

Area	WebSphere Application Server Version 6.0.2 (Client for Java Message Service on Java 2 Platform, Standard Edition with WebSphere Application Server)	WebSphere Application Server Version 7.0 (Thin Client for JMS with WebSphere Application Server)
Installation	Available for separate download and installation	Installed in the WebSphere Application Server or Application Client /runtimes directory
JMS jar file name	sibc.jms.jar	com.ibm.ws.sib.client.thin.jms_7.0.0.jar
Performing JNDI lookups of JMS resources	Requires optional sibc.jndi.jar (and ORB sibc.orb.jar for non-IBM JREs)	Requires Thin Client for EJB with WebSphere Application Server jar com.ibm.ws.ejb.thinclient_7.0.0.jar/ com.ibm.ws.ejb.thinclient.z_7.0.0.jar (and ORB com.ibm.ws.orb_7.0.0.jar for non-IBM JREs)
SSL configuration	Secure connections are configured using JRE global properties: **-Djavax.net.ssl.keyStore** **-Djavax.net.ssl.keyStorePassword** **-Djavax.net.ssl.trustStore** **-Djavax.net.ssl.trustStorePassword**	Two approaches to configuring secure connections. The first approach uses JRE global properties: **-Djavax.net.ssl.keyStore** **-Djavax.net.ssl.keyStorePassword** **-Djavax.net.ssl.trustStore** **-Djavax.net.ssl.trustStorePassword** The second approach is to specify security settings that are specific to Thin Client for JMS with WebSphere Application Server connections: **-Dcom.ibm.ssl.keyStoreType** **-Dcom.ibm.ssl.keyStore** **-Dcom.ibm.ssl.keyManager** **-Dcom.ibm.ssl.trustManager** **-Dcom.ibm.ssl.keyStorePassword** **-Dcom.ibm.ssl.protocol** **-Dcom.ibm.ssl.contextProvider** **-Dcom.ibm.ws.sib.jsseProvider**
Minimum JRE level	1.4.2	1.5
Enable trace (set trace specification)	**-Dcom.ibm.ws.sib.client.traceSetting**	**-Dcom.ibm.ejs.ras.lite.traceSpecification**
Set trace filename	**-Dcom.ibm.ws.sib.client.traceFile**	**-Dcom.ibm.ejs.ras.lite.traceFileName**
Set max trace file size	Not applicable	**-Dcom.ibm.ejs.ras.lite.maxFileSize**
Set max number of trace files	Not applicable	**-Dcom.ibm.ejs.ras.lite.maxFiles**
Set trace format	Not applicable	**-Dcom.ibm.ejs.ras.lite.traceFormat**

Table 16. Migration from WebSphere Application Server Version 6.0.2 to WebSphere Application Server Version 7.0 (continued)

Area	WebSphere Application Server Version 6.0.2 (Client for Java Message Service on Java 2 Platform, Standard Edition with WebSphere Application Server)	WebSphere Application Server Version 7.0 (Thin Client for JMS with WebSphere Application Server)
Using alternative trace properties file	**-DtraceSettingsFile**	**-DtraceSettingsFile** Thin Client for JMS with WebSphere Application Server supports new options described in “Trace user interface for standalone clients” on page 1664
Enable and specify FFDC filename	**-DffdcLogFile**	**-Dcom.ibm.ejs.ras.lite.ffdcLogFile**
NLS support	Install time option for NLS support	Non-English versions are available in sibc.nls.jar

Using JMS resources with the Thin Client for JMS with WebSphere Application Server:

Suitable JMS connection factories and references to JMS queues or topics may be obtained programmatically without using JNDI. Alternatively, full JNDI support may be obtained from the Thin Client for EJB with WebSphere Application Server.

- To obtain suitable connection factories programmatically, without using JNDI, use code similar to that shown in the following example:

  ```
  import com.ibm.websphere.sib.api.jms.*;
  ...
  JmsConnectionFactory jmsCF =
    JmsFactoryFactory.getInstance().createQueueConnectionFactory();
  jmsCF.setBusName("myBus");
  jmsCF.setProviderEndpoints("1.2.3.4");
  ```

  To obtain a suitable reference to a JMS queue or topic programmatically, use code similar to that shown in the following example:

  ```
  JmsQueue jmsQ = JmsFactoryFactory.getInstance().createQueue("myQueue");
  ```

  For further information, see the JmsFactoryFactory class API documentation available with WebSphere Application Server.
- To obtain full JNDI support from the Thin Client for EJB with WebSphere Application Server:
  1. Include the /runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar file in the compile and runtime classpaths for your enterprise application as described in “Installing and configuring the Thin Client for JMS with WebSphere Application Server” on page 1659.
  2. Use the following code to create a suitable Initial Context, substituting the server IP address and port as appropriate:

     ```
     import javax.naming.*;
     ...
     Properties env = new Properties();
     env.put(Context.PROVIDER_URL,"iiop:
       //<server IP address>:<server bootstrap address port>");
     env.put(Context.INITIAL_CONTEXT_FACTORY,
       "com.ibm.websphere.naming.WsnInitialContextFactory");
     InitialContext ctx = new InitialContext(env);
     ```

     In certain situations, for example when running with a Sun JRE, an additional ORB jar is also required. For additional information about when this jar is required, see Running the IBM Thin Client for Enterprise JavaBeans (EJB).

Obtaining WebSphere MQ JMS resources in the thin client environment:

A standalone Java SE JMS thin client application that connects to an external WebSphere MQ queue manager can get administratively-created WebSphere MQ messaging provider JMS resources from the WebSphere Application Server Java Naming and Directory Interface (JNDI) namespace.

1. To obtain WebSphere MQ messaging provider JMS resources from the WebSphere Application Server JNDI namespace in the thin client environment, include the following jar files in the runtime classpath of your application:
   - A copy of the /runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar.
   - A copy of the /runtimes/com.ibm.ws.messagingClient.jar.
   - WebSphere MQ client jar files which must be obtained from the WebSphere MQ product.
2. Use the following code to create a suitable Initial Context, substituting the server IP address and port as appropriate:

```
import javax.naming.*;
...
Properties env = new Properties();
env.put(Context.PROVIDER_URL,"iiop:
  //<server IP address>:<server bootstrap address port>");
env.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.ibm.websphere.naming.WsnInitialContextFactory");
InitialContext ctx = new InitialContext(env);
```

   In certain situations, for example when running with a Sun JRE, an additional ORB jar is also required. For additional information about when this jar is required, see the Thin Client for EJB with WebSphere Application Server information.

Running with other standalone clients:

The WebSphere Application Server standalone clients are the clients found in the %WAS_HOME%/runtimes directory. You can run standalone clients either on their own, or in conjunction with one or more other standalone clients.

Before you begin

When you are executing two or more of the following WebSphere Application Server standalone clients together, you must obtain all the clients that you are using from the same WebSphere Application Server installation or service refresh.

- Thin Client for JMS with WebSphere Application Server
- Thin Client for EJB with WebSphere Application Server
- Thin Client for JAX-WS with WebSphere Application Server
- Thin Client for JAX-RPC with WebSphere Application Server

This is because the mixing of standalone clients from different builds could result in unpredictable behavior since different builds might contain slightly different versions of the same class. The `CWSJE0001E` error message appears if the in-built consistency checker detects that you are trying to run clients with different build levels together.

Securing client and resource adapter connections

There are two approaches to configuring Secure Sockets Layer (SSL) for the Thin Client for JMS with WebSphere Application Server and the Resource Adapter for JMS with WebSphere Application Server. The global configuration approach affects all standalone outbound connections from the process, and the private approach applies only to client or resource adapter connections from the process.

About this task

The Thin Client for JMS with WebSphere Application Server and the Resource Adapter for JMS with WebSphere Application Server use the standard Java Secure Socket Extension (JSSE) that all supported JREs provide for making Secure Sockets Layer (SSL) connections. For information about JSSE, see the JSSE documentation.

The global configuration approach uses JRE global properties and affects all outbound SSL connections that your application initiates. For a JRE configured to use SSL connections to connect to WebSphere Application Server, you typically need to set the following javax.net.ssl system properties:

```
-Djavax.net.ssl.keyStore=key.p12
-Djavax.net.ssl.keyStorePassword={xor}Lz4sLCgwLTs=
-Djavax.net.ssl.trustStore=trust.p12
-Djavax.net.ssl.trustStorePassword={xor}PSo4LSov
```

The private configuration approach allows you to specify security settings that are specific to the Thin Client for JMS with WebSphere Application Server or the Resource Adapter for JMS with WebSphere Application Server connections. You can configure the com.ibm.ws.sib.client.ssl.properties system property to specify the location of an IBM SSL properties file. If this system property is not configured, an attempt is made load the properties file from the classpath instead.

The client obtains the value that it uses for any particular SSL property as follows:

- If the property has a value defined in the properties file containing the IBM SSL properties, the client uses this value.
- There is no value for the property in the properties file, the client next looks for a suitable property in the associated JRE system properties.
- If there is no suitable javax.net.ssl property, the client uses the default value.

The table below summarizes the IBM SSL property keys that can be configured inside the IBM SSL properties file, and the corresponding javax.net.sll.* system property keys and default values.

Table 17. IBM SSL property values and corresponding JRE global property and default values

IBM SSL property	JRE global property	Default value
com.ibm.ssl.keyStoreType	javax.net.ssl.keyStoreType	JKS
com.ibm.ssl.keyStore	javax.net.ssl.keyStore	None
com.ibm.ssl.keyManager	javax.net.ssl.keyStoreProvider	IbmX509
com.ibm.ssl.trustManager	javax.net.ssl.trustStoreProvider	IbmX509
com.ibm.ssl.keyStorePassword	javax.net.ssl.keyStorePassword	None
com.ibm.ssl.protocol	None	SSL
com.ibm.ssl.contextProvider	None	IBMJSSE2
com.ibm.ws.sib.jsseProvider	None	com.ibm.jsse2.IBMJSSEProvider2
com.ibm.ssl.trustStore	javax.net.ssl.trustStore	None
com.ibm.ssl.trustStoreType	javax.net.ssl.trustStoreType	JKS
com.ibm.ssl.trustStorePassword	javax.net.ssl.trustStorePassword	None

For example, you might create an ssl.properties file that contains the following properties and values:

```
com.ibm.ssl.keyStore=/thinclient/key.p12
com.ibm.ssl.keyStoreType=PKCS12
com.ibm.ssl.keyStorePassword=WebAS
com.ibm.ssl.trustStore=/thinclient/trust.p12
com.ibm.ssl.trustStoreType=PKCS12
com.ibm.ssl.trustStorePassword=WebAS
```

You can use the PropFilePasswordEncoder tool in the WebSphere Application Server bin directory to encode passwords stored in plain text property files. For further information see Encoding passwords in files.

Note:

1. SSL connections from SUN JREs using the Thin Client for JMS with WebSphere Application Server cannot use the default WebSphere Application Server PKCS12 key and trust stores. If you are running the client securely from SUN JREs, you must first extract the certificates from the trust store using an IBM software development kit (SDK). You can then import these certificates into a keystore that the Sun JRE can recognize correctly, such as a JKS keystore.
2. SSL connections are not supported using the IBM JRE shipped with WebSphere Application Server - a non-WebSphere Application Server installed JRE must be used.

1. Obtain the necessary key and trust store files.
2. Set the javax.net.ssl system properties required for the global configuration approach.
3. For the private configuration approach, use the com.ibm.ws.sib.client.ssl.properties system property to specify the file from which the SSL properties are to be loaded, as shown in the following example:

   ```
   -Dcom.ibm.ws.sib.client.ssl.properties=c:/ssl.properties
   ```

Adding tracing and logging for standalone clients

You can add tracing and logging to help analyze performance and diagnose problems.

About this task

This information applies to the following WebSphere Application Server standalone clients:

- Thin Client for JMS with WebSphere Application Server
- Thin Client for EJB with WebSphere Application Server
- Thin Client for JAX-WS with WebSphere Application Server
- Thin Client for JAX-RPC with WebSphere Application Server
- To enable trace, use either a long form or short form system property.

  Note: Trace settings are determined from the system property values the first time that a WebSphere Application Server client is called. The trace settings are then fixed. Therefore, any subsequent changes to the system property settings do not change the trace settings that the WebSphere Application Server client uses.
- To enable First Failure Data Capture (FFDC), use either a long or short form system property.

  Note: FFDC settings are determined from the system property values the first time that a WebSphere Application Server client performs an FFDC. The FFDC settings are then fixed. Therefore, any subsequent changes to the system property settings do not change the FFDC settings that the WebSphere Application Server client uses.

Trace user interface for standalone clients:

To enable trace, you can either use a long form or a short form system property.

Long form system properties

The long form system property takes priority over the short form and uses system properties that are unique to WebSphere Application Server.

Table 18. Long form system properties

Property	Description
com.ibm.ejs.ras.lite.traceSpecification	The trace specification string

Table 18. Long form system properties (continued)

Property	Description
com.ibm.ejs.ras.lite.traceFileName	The trace destination (<*file*>, stdout, stderr, java,util.logging)
com.ibm.ejs.ras.lite.maxFileSize	The maximum trace file size in mb (if the trace destination is a file)
com.ibm.ejs.ras.lite.maxFiles	The maximum number of trace files kept (if the trace destination is a file)
com.ibm.ejs.ras.lite.traceFormat	The trace output format, which can be either basic or advanced (the default is basic)

Long form example:

```
-Dcom.ibm.ejs.ras.lite.traceSpecification=SIB*=all
-Dcom.ibm.ejs.ras.lite.traceFileName=c:/trace.log
-Dcom.ibm.ejs.ras.lite.maxFileSize=20
-Dcom.ibm.ejs.ras.lite.maxFiles=8
```

Short form system property

The short form uses a system property which is compatible with existing WebSphere Application Server clients but which, since this property is unqualified, may clash with other third party technologies that are running in the same Java runtime environment (JRE).

The short form system property is:

```
traceSettingsFile
```

This property must specify a loadable properties file that can contain the following properties:

Table 19. Properties in loadable system properties file

Property	Header
traceFileName	The trace destination (file, stdout, stderr, java,util.logging)
maxFilesSize	The maximum trace file size in mb (if the trace destination is a file)
maxFiles	The maximum number of trace files kept (if the trace destination is a file)
<traceSpec>	The trace specification
traceFormat	The trace output format, which can be either basic or advanced (the default is basic)

The following example shows how to use the short form system property:

```
SIBTrm=all:SIBMfp=all
traceFileName=c:/trace.log
```

Special meanings for trace file name values

Some trace file name values have a special meaning:

- stdout - causes trace records to be written to stdout
- stderr - causes trace records to be written to stderr
- java.util.logging - causes trace records to be written to java.util.logging

Using any other name causes the trace records to be written to a file of that name.

First Failure Data Capture user interface for standalone clients:

To enable First Failure Data Capture (FFDC) output, you can either use a long or short form system property.

Long form system property

The long form takes priority over the short form and uses a system property that is unique to WebSphere Application Server to specify the FFDC dump file name. This property is:

```
com.ibm.ejs.ras.lite.ffdcLogFile
```

The following example shows how to use the long form system property to enable FFDC output:

```
-Dcom.ibm.ejs.ras.lite.ffdcLogFile=c:\ffdc.log
```

Short form system property

The short form system property is:

```
ffdcLogFile
```

The following example shows how to use the short form system property to enable FFDC output:

```
-DffdcLogFile=c:\ffdc.log
```

Using JMS from a third party application server

The Resource Adapter for JMS with WebSphere Application Server provides first class connectivity to service integration resources from the third party application server on which it is deployed.

About this task

The Resource Adapter for JMS with WebSphere Application Server is designed to be deployed into third party application servers, that is, into non-WebSphere Application Server application servers, that support Java EE Connector Architecture (JCA) 1. 5 and are Java 2 Platform, Enterprise Edition (J2EE) 1.4 compliant. The standalone resource adapter provides these third party application servers with full connectivity to service integration resources running inside WebSphere Application Server Version 6.0.2 and later.

Supported third party application servers are:

- WebSphere Application Server Community Edition Version 2.0 and later
- Apache Geronimo v2.0 and later
- JBoss Application Server v4.0.5 and later

Note:

- The resource adapter does not support the unmanaged JCA environment. In an unmanaged environment use the Thin Client for JMS with WebSphere Application Server.
- If you want to use XA connections to a WebSphere Application Server Version 6.0.2 application server, contact IBM Support to obtain a required service update.

Deploying the Resource Adapter for JMS with WebSphere Application Server

To provide connections to service integration resources running inside WebSphere Application Server, the Resource Adapter for JMS with WebSphere Application Server must be installed into the third party application server.

Before you begin

The Resource Adapter for JMS with WebSphere Application Server requires JRE 1.5 or later. The resource adapter is called sibc.jmsra.rar and is available from the following runtime directories:

- WebSphere Application Server install
- Application Client for WebSphere Application Server install

Before starting the deployment of the resource adapter, you must first obtain the following information from the WebSphere Application Server administrator:

- Bus name
- Endpoint provider address
- Target transport chain
- Messaging engine name
- Any other required connection and destination properties
- One or more destination names

The general approach to deploying the resource adapter is to write a deployment XML file to configure the required and optional properties for the JMS connection factory and JMS resources that will be accessed, and then deploy the resource adapter using the deployment XML file. The way in which you perform the installation depends on the particular application server that you are using. Before starting this task, see the documentation specific to your application server for information about how to install and use a JMS resource adapter RAR file.

About this task

An enterprise application that looks up a Resource Adapter for JMS with WebSphere Application Server connection factory in the local Java Naming and Directory Interface (JNDI) repository can access service integration resources through the resource adapter, provided that the required messaging engine is available in WebSphere Application Server. All outbound connections must access all queues and topics using Queue or Topic resources. These resources are configured using your particular application server configuration mechanism when the resource adapter is deployed.

The Resource Adapter for JMS with WebSphere Application Server supports full two-phase XA transactional connections (except under the JBoss Application Server) but can also be run using local transactions or no transaction connections.

Multiple deployments of the resource adapter are possible.

1. To deploy an outbound JMS resource on the Resource Adapter for JMS with WebSphere Application Server, use your particular application server configuration mechanism to configure the following service integration bus properties:
   - Bus name
   - Provider endpoints
2. If you want to use XA resources over a Resource Adapter for JMS with WebSphere Application Server connection, use your particular application server configuration mechanism to configure the following additional service integration bus properties:
   - Target type must be set to ″ME″
   - Target significance must be set to ″Required″
   - Target must be set to the name of the required messaging engine

   These properties permit the recovery of indoubt transactions, should this be necessary. For further information about indoubt transactions, see Resolving indoubt transactions.

See “Resource Adapter for JMS with WebSphere Application Server configuration properties” for a description of these property names and other properties that may also be configured.

Results

Subsequent usage of the resource adapter is in accordance with the Java EE programming specifications. That is, any enterprise bean or message-driven bean may obtain a Resource Adapter for JMS with WebSphere Application Server connection factory or use an activation specification to connect to a service integration messaging engine. Message-driven beans behave in just the same way as they would in any other Java EE environment.

What to do next

You can turn trace and First Failure Data Capture (FFDC) on for the resource adapter in the same way as for the Thin Client for JMS with WebSphere Application Server. For further information, see “Adding tracing and logging for standalone clients” on page 1664.

You can configure secure connections by configuring connection factories that require a secure bootstrap and/or connection transport chain in the same way as for the Thin Client for JMS with WebSphere Application Server. For further information, see “Securing client and resource adapter connections” on page 1662.

Resource Adapter for JMS with WebSphere Application Server configuration properties:

As part of deploying Resource Adapter for JMS with WebSphere Application Server, you must configure a set of JMS resources which the deployed resource adapter instance will support.

The following tables list the JMS properties and their values.

Note: Not all properties available to applications running inside the WebSphere Application Server environment are available in third party environments. Some properties have no meaning outside of the WebSphere Application Server environment and some have no meaning for remotely connected clients.

Table 20. Connection factory properties

Property name	Description	Permitted values	Default
BusName	The name of the service integration bus to connect to.		
ClientID	The JMS client identifier needed for durable topic subscriptions on all connections created using this connection factory.		
UserName			
Password			
NonPersistentMapping	The reliability applied to nonpersistent JMS messages sent using this connection factory.	BestEffortNonPersistent, ExpressNonPersistent, ReliableNonPersistent	ExpressNonPersistent
PersistentMapping	The reliability applied to persistent JMS messages sent using this connection factory.	ReliablePersistent, AssuredPersistent	ReliablePersistent

Table 20. Connection factory properties (continued)

Property name	Description	Permitted values	Default
DurableSubscriptionHome	The name of the messaging engine used to store messages delivered to durable subscriptions for objects created from this JMS connection factory.		
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, Default	Default
Target	The name of a target that identifies a group of messaging engines. Specify the type of target using the Target type property.		
TargetType	The type of target named in the Target property.	BusMember, Custom, ME	BusMember
TargetSignificance	The significance of the target group.	Required, Preferred	Required
TargetTransportChain	The name of the protocol that resolves to a group of messaging engines.		
ProviderEndpoints	The list of comma separated endpoints used to connect to a bootstrap server.		
ConnectionProximity	The proximity of messaging engines that can accept connection requests, in relation to the bootstrap messaging engine.	Server, Cluster, Host, Bus	Bus
TemporaryQueueNamePrefix	The prefix of up to twelve characters used for names of temporary queues created by applications that use this connection factory.		
TemporaryTopicNamePrefix	The prefix used at the start of temporary topics created by applications using this connection factory.		
ShareDurableSubscriptions	Controls whether or not durable subscriptions are shared across connections with members of a server cluster.	InCluster, AlwaysShared, NeverShared	InCluster (always resolves to AlwaysOff as the client is always outside of a WebSphere Application Server clustered server)
ProducerDoesNotModify PayloadAfterSet	When enabled, Object or Bytes messages sent by a message producing application that has connected to the bus using this connection factory will not have their data copied when set and the system will only serialize the message data when absolutely necessary. Applications sending such messages must not modify the data once it has been set into the message.	true, false	false

Table 21. Queue Connection factory properties

Property name	Description	Permitted values	Default
BusName	The name of the service integration bus to connect to.		
UserName			
Password			
NonPersistentMapping	The reliability applied to nonpersistent JMS messages sent using this connection factory.	BestEffortNonPersistent, ExpressNonPersistent, ReliableNonPersistent	ExpressNonPersistent
PersistentMapping	The reliability applied to persistent JMS messages sent using this connection factory.	ReliablePersistent, AssuredPersistent	ReliablePersistent
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, Default	Default
Target	The name of a target that identifies a group of messaging engines. Specify the type of target using the Target type property.		
TargetType	The type of target named in the Target property.	BusMember, Custom, ME	BusMember
TargetSignificance	The significance of the target group.	Required, Preferred	Required
TargetTransportChain	The name of the protocol that resolves to a group of messaging engines.		
ProviderEndpoints	The list of comma separated endpoints used to connect to a bootstrap server.		
ConnectionProximity	The proximity of messaging engines that can accept connection requests, in relation to the bootstrap messaging engine.	Server, Cluster, Host, Bus	Bus
TemporaryQueueNamePrefix	The prefix of up to twelve characters used for names of temporary queues created by applications that use this connection factory.		
ProducerDoesNotModify PayloadAfterSet	When enabled, Object or Bytes messages sent by a message producing application that has connected to the bus using this connection factory will not have their data copied when set and the system will only serialize the message data when absolutely necessary. Applications sending such messages must not modify the data once it has been set into the message.	true, false	false

Table 22. Topic Connection factory properties

Property name	Description	Permitted values	Default
BusName	The name of the service integration bus to connect to.		
ClientID	The JMS client identifier needed for durable topic subscriptions on all connections created using this connection factory.		
UserName			
Password			
NonPersistentMapping	The reliability applied to nonpersistent JMS messages sent using this connection factory.	BestEffortNonPersistent, ExpressNonPersistent, ReliableNonPersistent	ExpressNonPersistent
PersistentMapping	The reliability applied to persistent JMS messages sent using this connection factory.	ReliablePersistent, AssuredPersistent	ReliablePersistent
DurableSubscriptionHome	The name of the messaging engine used to store messages delivered to durable subscriptions for objects created from this JMS connection factory.		
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, Default	Default
Target	The name of a target that identifies a group of messaging engines. Specify the type of target using the Target type property.		
TargetType	The type of target named in the Target property.	BusMember, Custom, ME	BusMember
TargetSignificance	The significance of the target group.	Required, Preferred	Required
TargetTransportChain	The name of the protocol that resolves to a group of messaging engines.		
ProviderEndpoints	The list of comma separated endpoints used to connect to a bootstrap server.		
ConnectionProximity	The proximity of messaging engines that can accept connection requests, in relation to the bootstrap messaging engine.	Server, Cluster, Host, Bus	Bus
TemporaryTopicNamePrefix	The prefix used at the start of temporary topics created by applications using this connection factory.		
ShareDurableSubscriptions	Controls whether or not durable subscriptions are shared across connections with members of a server cluster.	InCluster, AlwaysShared, NeverShared	InCluster (always resolves to AlwaysOff as the client is always outside of a WebSphere Application Server clustered server)

Table 22. Topic Connection factory properties (continued)

Property name	Description	Permitted values	Default
ProducerDoesNotModify PayloadAfterSet	When enabled, Object or Bytes messages sent by a message producing application that has connected to the bus using this connection factory will not have their data copied when set and the system will only serialize the message data when absolutely necessary. Applications sending such messages must not modify the data once it has been set into the message.	true, false	false

Table 23. Queue properties

Property name	Description	Permitted values	Default
QueueName	The name of the associated queue on the service integration bus.		
DeliveryMode	The delivery mode for messages sent to this destination. This controls the persistence of messages on this destination.	Application, Persistent, NonPersistent	
TimeToLive	The default length of time in milliseconds from its dispatch time that a message sent to this destination should be kept by the system.		
Priority	The relative priority for messages sent to this destination, in the range 0 to 9, with 0 as the lowest priority and 9 as the highest priority.		
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, AsConnection, Default	AsConnection
BusName	The name of the service integration bus to connect to.		
ScopeToLocalQP	Sets whether the service integration bus queue destination identified by this Queue is dynamically scoped to a single queue point if one exists on the messaging engine that the application is connected to.	On, Off	Off
ProducerPreferLocal	Sets whether a MessageProducer for this Queue should prefer a locally connected queue point of the service integration bus queue destination over any other queue points.	On, Off	On

Table 23. Queue properties (continued)

Property name	Description	Permitted values	Default
ProducerBind	Set whether messages sent by a single MessageProducer to this Queue will go to the same service integration bus queue point, or whether no such restriction will be applied, and different messages will be sent to different queue points.	On, Off	Off
GatherMessages	Set whether messages on all service integration bus queue points or only a single queue point are visible to MessageConsumers and QueueBrowsers using this Queue.	On, Off	Off

Table 24. Topic properties

Property name	Description	Permitted values	Default
TopicSpace	The name of the topic space that contains the topic, on the service integration bus defined by the BusName property.		Default.Topic.Space
TopicName	The name of the topic that this JMS topic is assigned to, in the topic space defined by the TopicSpace property		
DeliveryMode	The delivery mode for messages sent to this destination. This controls the persistence of messages on this destination.	Application, Persistent, NonPersistent	
TimeToLive	The default length of time in milliseconds from its dispatch time that a message sent to this destination should be kept by the system.		
Priority	The relative priority for messages sent to this destination, in the range 0 to 9, with 0 as the lowest priority and 9 as the highest priority.		
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, AsConnection, Default	AsConnection
BusName	The name of the service integration bus to connect to.		

Table 25. Activation configuration properties

Property name	Description	Permitted values	Default	Required/optional
Destination	The name of the destination on the service integration bus.			Required
ProviderEndpoints	The list of comma separated endpoints used to connect to a bootstrap server.			Required

Table 25. Activation configuration properties (continued)

Property name	Description	Permitted values	Default	Required/optional
DestinationType	Whether the message-driven bean uses a queue or topic destination.	javax.jms.Queue, javax.jms.Topic		Required
BusName	The name of the service integration bus to connect to.			Required
MessageSelector	The JMS message selector used to determine which messages the message-driven bean receives. The value is a string that is used to select a subset of the available messages. The syntax is based on a subset of the SQL 92 conditional expression syntax, as described in the JMS specification.			Optional
AcknowledgeMode	How the session acknowledges any messages it receives.	Auto-acknowledge, Dups-ok-acknowledge	Auto-acknowledge	Optional
SubscriptionDurability	Whether a JMS topic subscription is durable or nondurable.	Durable, Nondurable	Nondurable	Optional
SubscriptionName	The subscription name needed for durable topic subscriptions. Required field when using a durable topic subscription.			Optional
MaxBatchSize	The maximum number of messages received from the messaging engine in a single batch.	1 through 2147483647	1	Optional
MaxConcurrency	The maximum number of endpoints to which messages are delivered concurrently	1 through 2147483647	10	Optional
RetryInterval	The delay (in seconds) between attempts to connect to a messaging engine.	1 through 2147483647	30	Optional
UserName				Optional
Password				Optional
DurableSubscriptionHome	The name of the messaging engine used to store messages delivered to durable subscriptions for objects created from this JMS connection factory.			Optional

Table 25. Activation configuration properties (continued)

Property name	Description	Permitted values	Default	Required/optional
ShareDurableSubscriptions	Controls whether or not durable subscriptions are shared across connections with members of a server cluster.	InCluster, AlwaysShared, NeverShared	InCluster (always resolves to AlwaysOff as the client is always outside of a WebSphere Application Server clustered server)	Optional
ClientID	The JMS client identifier needed for durable topic subscriptions on all connections created using this connection factory.			Optional
TargetTransportChain	The name of the protocol that resolves to a group of messaging engines.			Optional
ReadAhead	Read ahead is an optimization that preemptively assigns messages to consumers. This improves the time taken to satisfy consumer requests.	AlwaysOn, AlwaysOff, Default	Default	Optional
Target	The name of a target that identifies a group of messaging engines. Specify the type of target using the Target type property.			Optional
TargetType	The type of target named in the Target property.	BusMember, Custom, ME	BusMember	Optional
TargetSignificance	This property specifies the significance of the target group.	Required, Preferred	Required	Optional
TopicSpace	The name of the topic space that contains the topic, on the service integration bus defined by the BusName property.		Default.Topic.Space	Optional
ForwarderDoesNotModify PayloadAfterSet	When enabled, Object/Bytes messages forwarded through this activation specification that have their payload modified will not have the data copied when it is set into the message and the system will only serialize the message data when absolutely necessary. Applications sending such messages must not modify the data once it has been set into the message.	true, false	false	Optional

Deploying inbound connections for the resource adapter

When deploying inbound connections, you must configure message-driven beans to use the Resource Adapter for JMS with WebSphere Application Server. A JMS activation configuration associated with one or more message-driven beans provides the configuration necessary for them to receive messages.

About this task

You can deploy message-driven beans within your application with a JMS activation configuration to access the Resource Adapter for JMS with WebSphere Application Server connection factories and destinations. When the message-driven bean is started, it uses the resource adapter to connect to the service integration bus, provided that the required messaging engine is available in WebSphere Application Server.

The Resource Adapter for JMS with WebSphere Application Server supports full two-phase XA transactional connections but it may also be run using local transactions or no transaction connections.

1. Configure the following properties for the activation configuration:
   - Destination
   - ProviderEndpoints
   - DestinationType
   - BusName

   The Destination property value is the name of the destination from which the message-driven bean will be receiving messages.

   See “Resource Adapter for JMS with WebSphere Application Server configuration properties” on page 1668 for a description of these property names and other properties that may also be configured.
2. Configure the following additional properties if the message-driven bean will be using XA resources over the Resource Adapter for JMS with WebSphere Application Server connection.
   - TargetType must be set to ″ME″
   - TargetSignificance must be set to ″Required″
   - Target value must be the name of the required ME

   See “Resource Adapter for JMS with WebSphere Application Server configuration properties” on page 1668 for a description of these property names and other properties that may also be configured.

   These properties permit the recovery of in-doubt transactions, should this be necessary. For further information about in-doubt transactions, see Resolving indoubt transactions.

Choosing a messaging provider

For messaging between application servers, perhaps with some interaction with a WebSphere MQ system, you can use the default messaging provider. To integrate WebSphere Application Server messaging into a predominantly WebSphere MQ network, you can use the WebSphere MQ messaging provider. You can also use a third-party messaging provider. To choose the provider that is best suited to your needs, consider what the application needs to do, and the business need for the provider to integrate well with your enterprise infrastructure.

About this task

Enterprise applications in WebSphere Application Server can use asynchronous messaging through services based on Java Message Service (JMS) messaging providers and their related messaging systems. These messaging providers conform to the JMS Version 1.1 specification.

You can configure any of the following messaging providers:

- The default messaging provider (which uses service integration as the provider)
- The WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider)

- A third-party messaging provider (which uses another company's product as the provider)

The types of messaging providers that can be configured in WebSphere Application Server are not mutually exclusive:

- Different applications can use the same, or different, providers.
- One application can access multiple providers.

No one of these providers is necessarily better than another. The choice of provider depends on what your JMS application needs to do, and on other factors relating to your business environment and planned changes to that environment.

Note: For backwards compatibility with earlier releases, WebSphere Application Server also includes support for the "V5 default messaging provider". For more information, see "Maintaining Version 5 default messaging resources" on page 1699.

1. Determine the environment and application requirements.

   If you need to use a third-party messaging provider, or interoperate with WebSphere Application Server Version 5 resources, use the associated provider. For more information, see "Other ways of managing messaging" on page 1689.

   If your existing or planned messaging environment involves both WebSphere MQ and WebSphere Application Server systems, and it is not clear to you whether you should use the default messaging provider, the WebSphere MQ provider, or a mixture of the two, complete the task "Choosing messaging providers for a mixed environment" on page 1683.

2. Choose the messaging provider:
   - Choose the default messaging provider.

     If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is the natural choice because this provider is fully integrated with the WebSphere Application Server runtime environment. For more information, see "Default messaging provider" on page 1815. To configure and manage messaging with the default messaging provider, see Managing messaging with the default messaging provider.
   - Choose the WebSphere MQ messaging provider.

     If your business also uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominantly WebSphere MQ network, the WebSphere MQ messaging provider allows you to define resources for connecting directly to the queues in a WebSphere MQ system. For more information, see "WebSphere MQ messaging provider" on page 1816. To configure and manage messaging with the WebSphere MQ messaging provider, see "Managing messaging with the WebSphere MQ messaging provider" on page 1851.
   - Choose a third-party messaging provider.

     You can use any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

     Note:

     – To administer a third-party messaging provider, use the resource adaptor or client supplied by the third party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

     – To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the Java EE Connector Architecture (JCA) Version 1.5 specification.

To work with a third-party provider, see “Managing messaging with a third-party messaging provider” on page 1689.

JMS providers collection

In the administrative console page, to view this page click **Resources** → **JMS** → **JMS providers**

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS providers that are available to WebSphere applications. For each JMS provider in the list, the entry indicates the *scope* level at which JMS resource definitions are visible to applications. You can create the same type of JMS provider at different Scope settings, to offer JMS resources at different levels of visibility to applications.

If you want to manage existing JMS resource definitions, or create a new JMS resource definition, you can select the name of one of the JMS providers in the list.

If you want to define your own JMS provider, other than the default messaging provider or WebSphere MQ, select the Scope setting at which JMS resource definitions are to be visible for that provider, then click **New**.

General properties

Name

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Select JMS resource provider

The variable, *{0}*, indicates the type of JMS resource that you are creating.

You select the scope setting on an earlier page. The choice of JMS providers depends on the scope that you selected. You might see a choice like the following list:

- Default messaging provider.

  Select this option if you want to the type of JMS resource to be provided by a service integration bus, as part of WebSphere Application Server.
- My JMSprovider

  Select this option if you want the type of JMS resource to be provided by your own JMS provider; not the default messaging provider or WebSphere MQ. You assign the name, in this example “My JMSprovider”, when you define the JMS provider to WebSphere Application Server. You must have installed and configured your own JMS provider before applications can use the JMS resources.

- WebSphere MQ messaging provider

  Select this option if you want the type of JMS resource to be provided by WebSphere MQ. You must have installed and configured WebSphere MQ before applications can use the JMS resources.
- V5 default messaging provider

  Select this option if you want the type of JMS resource to be provided by a WebSphere Application Server Version 5 node.

Activation specification collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Activation specification**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS activation specifications that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use a JMS activation specification if you want to use a message-driven bean as a J2EE Connector Architecture (JCA) 1.5 resource, to act as a listener on the default messaging provider.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Connection factory collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory of the JMS provider named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.1 domain-independent interfaces (referred to as the “common interfaces” in the JMS specification).

This type of JMS connection factory can also be used by the domain-specific (queue and topic) interfaces, as used in JMS 1.0.2, so applications can still use those interfaces without the need for you to create a domain-specific connection factory, such as a queue connection factory.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Queue connection factory collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Queue connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS queue connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory for the JMS provider that is named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.0.2 queue-specific interfaces.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Queue collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Queue**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS queue destinations that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use topic destination administrative objects to manage JMS queues for the JMS provider that is named in the Provider column of the list. Connections to the queue are created by a connection factory (or queue connection factory) for that JMS provider.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Topic connection factory collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Topic connection factory**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS topic connection factories that are available to WebSphere applications at the scope indicated by the **Scope** field.

A JMS connection factory is used to create connections to JMS destinations. When an application needs a JMS connection, an instance can be created by the factory for the JMS provider that is named in the Provider column of the list.

This type of connection factory is for applications that use the JMS 1.0.2 topic-specific interfaces.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Topic collection

In the administrative console page, to view this page click **Resources** → **JMS** → **Topic**.

To browse or change the properties of a listed item, select its name in the list.

To act on one or more of the listed items, select the check boxes next to the names of the items that you want to act on, then use the buttons provided.

To change what entries are listed, or to change what information is shown for entries in the list, use the Filter settings.

This page lists the JMS topic destinations that are available to WebSphere applications at the scope indicated by the **Scope** field.

Use topic destination administrative objects to manage JMS topics for the JMS provider that is named in the Provider column of the list. Connections to the topic are created by a connection factory (or topic connection factory) for that JMS provider.

General properties

Name

Provider

Description

Scope The level to which this resource definition is visible; for example, the cell, node, cluster, or server level.

Buttons

New	Click this button to create a new JMS resource of this type.
Delete	Click this button to delete the selected items.

Choosing messaging providers for a mixed environment

If your existing or planned messaging environment involves both WebSphere MQ and WebSphere Application Server systems, choose between the default messaging provider, the WebSphere MQ messaging provider, or a mixture of the two, by considering your messaging requirements, your business environment, and the needs of each messaging application.

About this task

For messaging between application servers, with interaction with a WebSphere MQ system, you can use the default messaging provider or the WebSphere MQ provider. Neither provider is necessarily better than the other. The choice of providers depends primarily on factors relating to your business environment and planned changes to that environment, and also on what each JMS application needs to do. Moreover, these two types of messaging providers are not mutually exclusive:

- You can configure both types of providers within one cell.
- Different applications can use the same, or different, providers.

Factors relating to your business environment include the following:

- Messaging requirements
- Existing skill set
- Existing messaging infrastructure
- Planned changes to that infrastructure

Configuring and managing your messaging infrastructure is simpler if you use just one provider. If your messaging is primarily in WebSphere MQ, you should probably choose the WebSphere MQ messaging provider. Similarly, if your messaging is primarily in WebSphere Application Server, you should probably choose the default messaging provider.

If your business environment does not clearly indicate that you should use just one provider, then you should consider using a mixture of the two, and choosing the most appropriate messaging provider for each application. A useful way of doing this is to identify the types of destinations (service integration bus, or WebSphere MQ queue or topic) that the application is using. If the application uses only bus destinations, the natural choice is to use the default messaging provider (solution ″DMP″). If the application needs to communicate with one or more WebSphere MQ destinations, you can choose any of the following solutions depending upon your business environment, usage scenarios, and system topologies:

- Use the WebSphere MQ messaging provider (solution ″MQP″).
- Use the default messaging provider to integrate a WebSphere MQ server (a WebSphere MQ queue manager or queue-sharing group) as a bus member (solution ″DMP interop bus member″).
- Use the default messaging provider to integrate a WebSphere MQ network as a foreign bus, using WebSphere MQ links (solution ″DMP interop, foreign bus″).

For more information about these solutions, see Overview of interoperation with WebSphere MQ.

To help you choose between these solutions, several of the following steps contain tables in which each row represents a business or system requirement, and asterisks (*) indicate the solutions that are likely to be most effective for meeting the requirement. These tables are designed to provide general guidance,

rather than to identify precisely a “right” solution. Most requirements have multiple possible solutions, and the absence of an asterisk does not necessarily mean that you cannot use that solution. To derive best guidance from using each of these tables:

- Focus on the rows that reflect your most important requirements.
- For all the rows that you consider, count the number of asterisks for each solution.

The solutions with the largest number of asterisks are likely to be the most effective.

1. If you have limited experience of WebSphere MQ or WebSphere Application Server, and are trying to decide which product best meets your messaging needs, see “Service integration and WebSphere MQ messaging - a comparison” on page 1688.

   Note: Whichever of these products you choose as the main focus for your messaging, you can still use either the default messaging provider or the WebSphere MQ provider for interoperation between the products.

2. Consider your business environment, to see if you can use just one provider.

   In deciding which provider to use, consider the following constraints:

   - The current and future messaging requirements
   - The existing messaging infrastructure
   - The skill set that you have in your organization

   If the majority of your messaging is today performed in WebSphere MQ, continue with that approach and configure WebSphere MQ as an external JMS provider (that is, use the WebSphere MQ messaging provider) in WebSphere Application Server. If the JMS requirements of your WebSphere Application Server applications are limited, it is debatable whether using a service integration bus for those applications gives sufficient benefit.

   If you have messaging applications in WebSphere Application Server that have no requirement to interoperate with your WebSphere MQ network, use the default messaging provider (the service integration bus). If your WebSphere Application Server messaging requirements demand a tighter integration with WebSphere Application Server, the service integration bus provides the following benefits:

   - Integrated administration
   - WebSphere Application Server high availability capabilities
   - WebSphere Application Server scalability

   If you choose to use the default messaging provider to interoperate between service integration and WebSphere MQ, be aware that there is an added cost involved in converting messages between service integration format and WebSphere MQ format.

   Also consider the following messaging scenarios:

   - A large installed backbone of WebSphere MQ queue managers, perhaps with WebSphere Message Broker.

     If you want to use WebSphere Application Server to run a newly introduced messaging application, you can deploy a WebSphere Application Server (JMS) messaging application that will exchange messages with an existing application that uses a WebSphere MQ queue or topic.

   - A WebSphere Application Server installation, perhaps with existing Web and enterprise applications, but no WebSphere Application Server messaging application.

     If you have no existing messaging infrastructure, you can deploy a WebSphere Application Server (JMS) messaging application to exchange messages with an existing WebSphere Application Server messaging application that uses a service integration bus destination.

   - An infrastructure that uses WebSphere Application Server to connect WebSphere Application Server messaging applications.

     Introduce WebSphere Application Server (JMS) messaging between a pair of WebSphere Application Server applications.

- An infrastructure that includes both WebSphere MQ and service integration buses. This could be the result of a merger, or because the message traffic tends to be from WebSphere Application Server to WebSphere Application Server, or from WebSphere MQ to WebSphere MQ, but not typically between WebSphere Application Server and MQ.

  Deploy a WebSphere Application Server (JMS) messaging application to exchange messages with an application that uses a WebSphere MQ queue or topic.
- A WebSphere Process Server or WebSphere Enterprise Service Bus infrastructure, which uses Service Component Architecture (SCA).

  You can choose either a WebSphere MQ or a service integration bus binding for your SCA components.

3. If your business environment does not clearly indicate that you should use just one messaging provider, use a mixture of the two and choose the most appropriate provider for each application, based upon the destination types that the application uses.

   The application might need to exchange messages with existing partner applications or services that use one or more known destinations of known type. Alternatively, the partner applications or services might not yet be deployed and the choice of destination type might still be open, in which case the solution architect needs to decide how best to connect the applications or services together.

   If the application uses multiple destinations, there are four possible outcomes:
   - The application uses only bus destinations.
   - The application uses only WebSphere MQ destinations.
   - The application uses a mixture of bus and WebSphere MQ destinations.
   - The destination types are not yet known.

   Note: If there is no clear business or technical reason why the application uses WebSphere MQ destinations rather than bus destinations, and the partner application is also a WebSphere Application Server JMS application, consider migrating the existing destinations to service integration so that the application uses only bus destinations.
4. If the application uses only bus destinations, configure the application and its JMS resources to use the default messaging provider.
5. If the application uses only WebSphere MQ destinations (queues or topics), use the following checklist to determine which provider solution to use.

Table 26. Provider checklist for an application that uses only WebSphere MQ destinations.

Question:	MQP	DMP interop, bus member	DMP interop, foreign bus
Is performance critical? (If so, use WebSphere MQ directly, rather than perform message conversion.)	*		
Does the application need to send or receive large messages (that is, messages > 500k.)?	*		
Is location transparency desirable for simplifying programming and deployment of applications?		*	*
Does the application need to consume from a WebSphere MQ queue, the configuration of which is fixed? (That is, the queue cannot be moved to service integration and you do not want to deploy a push-style WebSphere MQ application to send messages to a bus destination.)	*	*	

Table 26. Provider checklist for an application that uses only WebSphere MQ destinations. (continued)

Question:	MQP	DMP interop, bus member	DMP interop, foreign bus
Is the partner application a JMS application that will run outside WebSphere Application Server, as a bus or WebSphere MQ client? (Do not mix service integration and WebSphere MQ unless you have to do so; a pure WebSphere MQ or service integration solution is simpler and avoids the cost of converting messages between service integration and WebSphere MQ formats.)	*		
Is the partner application a non-JMS (non-WebSphere Application Server) application? (Wherever possible choose a pure WebSphere MQ or service integration solution. Use the MQI WebSphere MQ client, or the XMS WebSphere MQ client, or the XMS bus client depending on your API preference.)	*		
Do you prefer traffic passing between your WebSphere MQ network and WebSphere Application Server applications to be funneled into a single long-running connection?			*
Do you want to use the high availability features of WebSphere Application Server?			*
Is XA 2pc needed between the application and a WebSphere MQ queue sharing group?		*	*
Is XA 2pc needed between the application and a WebSphere MQ cluster?		*	

6. If the application uses a mixture of bus and WebSphere MQ destinations, for example consuming from service integration and sending to WebSphere MQ, then either of the default messaging provider interoperation models can support this by using a single connection factory or activation specification. Use the following checklist to help you decide between a bus member and a foreign bus solution.

Table 27. Provider checklist for an application that uses a mixture of bus and WebSphere MQ destinations.

Question:	DMP interop, bus member	DMP interop, foreign bus
Does the application need to consume from a WebSphere MQ shared queue?	*	
Is there a need to distribute work to a pool of WebSphere Application Server workers from a WebSphere MQ queue?	*	
Do you prefer traffic passing between your WebSphere MQ network and WebSphere Application Server applications to be funneled into a single long-running connection?		*
Do you need distributed WebSphere MQ in versions earlier than WebSphere Application Server Version 7 and WebSphere MQ Version 7?		*
Do you want store and forward capabilities to allow the application to continue to send messages when the WebSphere MQ queue manager is unavailable?		*
Do you prefer not to configure server connection channels? (This is because they open a port, which could be seen as a security risk.)		*
Do you prefer to define a server connection channel, rather than a pair of sender and receiver channels?	*	

7. If the destination types are not yet known, decide the relative priorities of the known concerns then use the following checklist to assess how well each of them is addressed by the possible provider solutions.

   The choice is really over what type of destinations this application should use. The destination types are not yet fixed, so any of the four solutions is possible, but as a general rule you should aim for solution “DMP” or “MQP”, because a pure WebSphere MQ or service integration solution is simpler

and avoids the cost of converting messages between service integration and WebSphere MQ formats.

Table 28. Provider checklist for an application for which the destination types are not yet known.

Question:	DMP	MQP	DMP interop, bus member	DMP interop, foreign bus
Do you have an existing base of strong skills in managing WebSphere MQ?		*	*	*
Do you want management of all messaging to be handled by the WebSphere MQ team?		*		
Do you have administrators skilled in WebSphere Application Server but not in WebSphere MQ?	*			
Do you want a messaging product with a large installed base (including references) and a wide choice of ISV tools?		*		
Are you reluctant to buy a separately licensed product in addition to WebSphere Application Server?	*			
Are you reluctant to install and manage a separate product in addition to WebSphere Application Server?	*			
Are you already using WebSphere Message Broker? (If so, you need WebSphere MQ anyway).		*	*	*
Are you using the WebSphere Enterprise Service Bus to mediate messages from, or deliver them to, a WebSphere MQ queue? (For example, using WebSphere Business Integration adapters, or connecting to a service provider such as CICS.)		*		
Does the application need to send or receive large messages (that is, messages > 500k.)?		*		
Is location transparency desirable for simplifying programming and deployment of applications?	*		*	*
Do the throughput requirements need multiple parallel channels or routes?		*	*	*
Does the application need to consume from a WebSphere MQ queue, the configuration of which is fixed? (that is, the queue cannot be moved to service integration and you don't want to deploy a push-style WebSphere MQ application to send messages to a bus destination.)	*	*	*	
Is the partner application a JMS application that will also run in WebSphere Application Server? (Service integration runs in the WebSphere Application Server application server. On distributed platforms that means it is in-process. On the z/OS platform it is in another region.)	*			
Is the partner application a JMS application that will run outside WebSphere Application Server, as a bus or WebSphere MQ client? (Do not mix service integration and WebSphere MQ unless you have to do so; a pure WebSphere MQ or service integration solution is simpler and avoids the cost of converting messages between service integration and WebSphere MQ formats.)	*	*		
Is the partner application a non-JMS (non-WebSphere Application Server) application? (Wherever possible choose a pure WebSphere MQ or service integration solution. Use the MQI WebSphere MQ client, or the XMS WebSphere MQ client, or the XMS bus client depending on your API preference.)	*	*		

Table 28. Provider checklist for an application for which the destination types are not yet known. (continued)

Question:	DMP	MQP	DMP interop, bus member	DMP interop, foreign bus
Is maintenance of strict message order important?	*			
Does the application require the flexibility and convenience of a WebSphere MQ cluster? (WebSphere MQ clustering makes administration simpler and provides selective parallelism of clustered queues. That is, instances of a clustered queue can be created on any (but not necessarily all) queue managers in the WebSphere MQ cluster. Messages sent to the clustered queue can be addressed to a specific instance of the queue, or allowed to select an instance dynamically based on workload management statistics. WebSphere Application Server clustering provides some of this flexibility, but you cannot create partitions of a bus destination on a subset of the messaging engines in a cluster bus member.)		*	*	*
Does the application need the level of high availability provided by WebSphere MQ for z/OS shared queues?		*	*	*
Do you want to use the high availability or scalability features of WebSphere Application Server clustering?	*		*	*

Service integration and WebSphere MQ messaging - a comparison

If you are not already an established user of either WebSphere Application Server or WebSphere MQ, and you are considering whether the service integration platform or WebSphere MQ better meets your messaging needs, use this table to compare the main features of the two platforms.

Table 29. Comparison of service integration and WebSphere MQ main features

service integration (the default messaging provider for WebSphere Application Server)	WebSphere MQ
Service integration is closely integrated with WebSphere Application Server, and is a natural fit if you are using the Java Platform, Enterprise Edition (Java EE).	WebSphere MQ can connect to almost anything. It provides a very heterogeneous environment .
Service integration supports multiple languages through XMS clients, and multiple platforms.	WebSphere MQ supports multiple languages and multiple platforms.
Service integration is a single process, pure Java implementation.	WebSphere MQ has many Independent Software Vendor (ISV) tools.
Service integration provides strong performance for both persistent and non-persistent messages for JMS.	WebSphere MQ supports JMS and non-JMS messaging interfaces, and provides strong performance for non-JMS applications.
Service integration is designed for a maximum message size of about 40 megabytes on a 32-bit operating system (subject to heap usage).	WebSphere MQ supports large message sizes up to about 100 megabytes.
Service integration is tightly integrated with some Web services implementations.	WebSphere MQ is a natural fit if you are using WebSphere Message Brokers.
Service integration messaging is included in a single administrative model for WebSphere Application Server, WebSphere Enterprise Service Bus and WebSphere Process Server.	WebSphere MQ can integrate existing infrastructure and applications (for example CICS).
Service integration bus clustering is integrated with WebSphere Application Server clustering for high availability and scalability.	WebSphere MQ clustering provides selective parallelism of clustered queues.

Note: The messaging platform that you choose for a given task does not necessarily determine which JMS messaging provider you should use. For example:

- If you need to handle large messages, you probably need to use the WebSphere MQ messaging provider.
- If you are using WebSphere Message Brokers, you can use either the default messaging provider or the WebSphere MQ messaging provider.

Other ways of managing messaging

For messaging between application servers, most requirements are best met by either the default messaging provider or the WebSphere MQ messaging provider. However, you can instead use a third-party messaging provider (that is, use another company's product as the provider). For backwards compatibility with earlier releases, there is also support for the V5 default messaging provider.

Before you begin

If you are not sure which provider combination is best suited to your needs, see "Types of messaging providers" on page 1814.

About this task

Enterprise applications in WebSphere Application Server can use asynchronous messaging through services based on Java Message Service (JMS) messaging providers and their related messaging systems. These messaging providers conform to the JMS Version 1.1 specification.

The choice of provider depends on what your JMS application needs to do, and on other factors relating to your business environment and planned changes to that environment.

- Choose a third-party messaging provider.

  You can configure any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

  To administer a third-party messaging provider, you use the resource adaptor or client supplied by the third party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

  To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the Java EE Connector Architecture (JCA) Version 1.5 specification.

  To work with a third-party provider, see "Managing messaging with a third-party messaging provider."
- Choose the (deprecated) V5 default messaging provider.

  This deprecated provider is identical to the WebSphere Application Server Version 5 default provider. Only the name has changed. It provides backwards compatibility that enables WebSphere Application Server Version 6 or later applications to connect to WebSphere Application Server Version 5 resources in a mixed cell. It also allows WebSphere Application Server Version 5 applications to connect to WebSphere Application Server Version 6 or later resources in a mixed cell. To configure and manage messaging to interoperate with WebSphere Application Server Version 5, see "Maintaining Version 5 default messaging resources" on page 1699.

Managing messaging with a third-party messaging provider

Enabling your messaging applications to use JMS resources provided by a third-party messaging provider other than WebSphere MQ.

Before you begin

For messaging between application servers, perhaps with some interaction with a WebSphere MQ system, you can use the default messaging provider. To integrate WebSphere Application Server messaging into a predominately WebSphere MQ network, you can use the WebSphere MQ messaging provider. You can also use a third-party messaging provider as described in this topic. To choose the provider that is best suited to your needs, see “Choosing a messaging provider” on page 1676.

About this task

You can install a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider.

WebSphere Application Server applications can use the JMS 1.1 interfaces or JMS 1.0.2 interfaces to access JMS resources provided by a third-party messaging provider, and you can use the administrative console to administer the JMS connection factories and destinations for the provider.

To use a third-party messaging provider with WebSphere Application Server, complete one or more of the following steps:

- Define a third-party messaging provider.
- List third-party JMS messaging resources.
- Configure JMS resources for a third-party messaging provider.

Defining a third-party messaging provider

Use this task to define a third-party messaging provider to WebSphere Application Server, for use instead of the default messaging provider or WebSphere MQ messaging provider.

Before you begin

Before you configure a third-party messaging provider, you might want to check whether your requirement can be met by the default messaging provider or the WebSphere MQ messaging provider that are supplied with WebSphere Application Server. To choose the provider that is best suited to your needs, see “Choosing a messaging provider” on page 1676.

About this task

You can configure any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

To administer a third-party messaging provider, you use the resource adaptor or client supplied by the third-party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the J2EE Connector Architecture (JCA) Version 1.5 specification.

To define a new third-party messaging provider to WebSphere Application Server, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources** → **JMS** → **JMS Providers** . The existing messaging providers are displayed, including the default messaging provider and the WebSphere MQ messaging provider.
2. To define a new third-party messaging provider, click **New** in the content pane. Otherwise, to change the definition of an existing messaging provider, click the name of the provider.

3. Specify the following required properties. You can specify other properties, as described in a later step.

   Name The name by which this messaging provider is known for administrative purposes within IBM WebSphere Application Server.

   External initial context factory
   The Java classname of the initial context factory for the JMS provider.

   External provider URL
   The JMS provider URL for external JNDI lookups.
4. Optional: Click **Apply**. This enables you to specify additional properties.
5. Optional: Specify other properties for the messaging provider.

   Under Additional Properties, you can use the **Custom Properties** link to specify custom properties for your initial context factory, in the form of standard javax.naming properties.
6. Click **OK**.
7. Save the changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

What to do next

You can now configure JMS resources for your messaging provider, as described in “Configuring JMS resources for a third-party messaging provider” on page 1697.

Listing third-party JMS messaging resources

Use this task with the WebSphere Application Server administrative console to display administrative lists of JMS resources provided by a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider.

About this task

You can use the WebSphere Application Server administrative console to display lists of the following types of JMS resources provided by a 3rd party messaging provider. You can use the panels displayed to select JMS resources to administer, or to create or delete JMS resources (where appropriate).

To display administrative lists of JMS resources for a third-party messaging provider, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
3. Click the name of the third-party messaging provider.
4. In the content pane, under Additional Resources, click the link for the type of JMS resource. For more information about the detailed settings displayed for each resource type, see the related reference topics.

JMS provider collection:

Use this panel to list JMS providers, or to select a JMS provider to view or change its configuration properties.

To view this administrative console page, click **Resources** → **JMS** → **JMS providers**

To view or change the properties of a JMS provider or its resources, select its name in the list displayed.

To define a new third-party messaging provider, click **New**.

To act on one or more of the JMS providers listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.
Name The name by which this JMS provider is known for administrative purposes.
Description
A description of this JMS provider for administrative purposes.

For related information about JMS messaging providers and asynchronous messaging, see
- "Types of messaging providers" on page 1814
- "Asynchronous messaging in WebSphere Application Server using JMS" on page 266

Third-party JMS connection factory collection:

The JMS connection factories configured for a third-party messaging provider for both point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS connection factories, or to select a connection factory to browse or change its configuration properties.

This panel shows a list of the JMS connection factories configured for a third-party messaging provider, with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS connection factory.
3. Under Additional Properties, click **Connection factories**.

To define a new JMS connection factory, click **New**.

To view or change the properties of a JMS connection factory, select its name in the list displayed.

To act on one or more of the JMS connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Third-party JMS connection factory settings:

Use this panel to browse or change the configuration properties of a JMS connection factory configured for use with a third-party messaging provider. These configuration properties control how connections are created to the JMS destinations on the provider.

A JMS connection factory is used to create connections to JMS destinations. The JMS connection factory is created by the associated JMS provider.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. If appropriate, in the content pane, change the scope of the third-party messaging provider.
3. In the content pane, click the name of the messaging provider that supports the JMS connection factory.
4. Under Additional Properties, click **Connection factories**.
5. Click the name of the JMS connection factory that you want to work with.

A JMS connection factory for a third-party messaging provider (that is, a provider other than the default, the V5 default or the WebSphere MQ messaging provider) has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated messaging provider.

Data type String

Type:

Whether this connection factory is for creating JMS queue destinations or JMS topic destinations.

Select one of the following options:

QUEUE
A JMS queue connection factory for point-to-point messaging.

TOPIC
A JMS topic connection factory for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the connection factory into the WebSphere Application Server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the connection factory into the name space of the third-party messaging provider.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Global Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum
Default Null

Range

ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to the messaging provider.

Third-party JMS destination collection:

The JMS destinations configured in an associated third-party messaging provider for point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS destinations, or to select a JMS destination to browse or change its configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging.

To define a new JMS destination, click **New**.

To view or change the properties of a JMS destination, select its name in the list displayed.

To act on one or more of the JMS destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Third-party JMS destination settings:

Use this panel to browse or change the configuration properties of the selected JMS destination for use with an associated third-party messaging provider.

A JMS destination is used to configure the properties of a JMS destination for the associated third-party messaging provider (that is, not the default messaging provider or the WebSphere MQ messaging provider). Connections to the JMS destination are created by the associated JMS connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging.
4. Click the name of the JMS destination that you want to work with.

A JMS destination for use with a third-party messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publish/subscribe).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String

Category:

A category used to classify or group this queue, for your WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Configuring JMS resources for a third-party messaging provider

Use the following tasks to configure the JMS connection factories and destinations for a third-party messaging provider (that is, a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider).

Before you begin

You only need to complete these tasks if your WebSphere Application Server environment uses a third-party messaging provider to support enterprise applications that use JMS. To enable use of a third-party messaging provider, you must have installed and configured the messaging provider, as described in Defining a third-party messaging provider.

About this task

To configure JMS resources for a third-party messaging provider, complete the following tasks:

- Configure a JMS connection factory for a third-party messaging provider.
- Configure a JMS destination for a third-party messaging provider.

Configuring a JMS connection factory for a third-party messaging provider:

Use this task to browse or change the properties of a JMS connection factory for use with a JMS messaging provider other than the default, V5 default or WebSphere MQ messaging providers.

About this task

To configure a JMS connection factory for use with a third-party messaging provider, use the administrative console to complete the following steps:

1. Display the third-party messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS Providers** .
2. Select the third-party provider for which you want to configure a connection factory.
3. Optional: Change the **Scope** setting to the level at which the connection factory is visible to applications.
4. In the content pane, under Additional Properties, click **Connection factories** This displays a table listing any existing JMS connection factories, with a summary of their properties.
5. To browse or change an existing JMS connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server.

      Type Select whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).

      JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the WebSphere Application Server name space.

      External JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the name space of the messaging provider.

   c. Click **Apply**. This defines the JMS connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the JMS connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS destination for a third-party messaging provider:

Use this task to browse or change the properties of a JMS destination for use with a third-party messaging provider (that is, a provider other than the default messaging provider or the WebSphere MQ messaging provider).

About this task

To configure a JMS destination for use with a third-party messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Click the name of the third-party messaging provider.
3. In the content pane, under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging. This displays a table listing any existing JMS destinations, with a summary of their properties.
4. To browse or change an existing JMS destination, click its name in the list. Otherwise, to create a new destination, complete the following steps:
   a. Click **New** in the content pane.

b. Specify the following required properties. You can specify other properties, as described in a later step.

Name The name by which this JMS destination is known for administrative purposes within WebSphere Application Server.

Type Select whether the destination is for JMS queues (QUEUE) or JMS topics (TOPIC).

JNDI Name
The JNDI name that is used to bind the JMS destination into the WebSphere Application Server name space.

External JNDI Name
The JNDI name that is used to bind the JMS destination into the name space of the messaging provider.

c. Click **Apply**. This defines the JMS destination to WebSphere Application Server, and enables you to browse or change additional properties.

5. Optional: Change properties for the JMS destination, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Maintaining Version 5 default messaging resources

This topic is the entry-point into a set of topics about maintaining messaging resources provided for WebSphere Application Server Version 5.1 applications by the default messaging provider.

About this task

JMS applications running on WebSphere Application Server Version 5.1 can use JMS resources provided by the default messaging provider in WebSphere Application Server Version 7. You can use the WebSphere Application Server administrative console to manage the JMS connection factories and destinations for WebSphere Application Server Version 5.1 applications. Such JMS resources are maintained as *V5 Default Messaging* resources.

V5 Default Messaging provides a JMS transport to a messaging engine of a service integration bus that supports the default messaging provider in WebSphere Application Server Version 7. The messaging engine emulates the service of a JMS server running on WebSphere Application Server Version 5.1.

You can also use the administrative console to manage a JMS server on a Version 5.1 node.

- List Version 5 default messaging resources.
- Configure Version 5 default JMS resources.
- Configure authorization security for a Version 5 default messaging provider.

Listing Version 5 default messaging resources

Use the WebSphere Application Server administrative console to list JMS resources for the Version 5 default messaging provider, for administrative purposes.

About this task

You use the WebSphere Application Server administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:

- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue

- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:

- Provider-centric navigation. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS providers** is provider-centric.
- Resource-centric navigation lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → **[resource]**, where [resource] is one of the resource types listed above is resource-centric.

You can use either of these navigation pathways to locate resources of any type.

- Using provider-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
  3. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
  4. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.
  5. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties
- Using resource-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.
  3. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties.

JMS provider settings:

Use this panel to view the configuration properties of a selected JMS provider. You cannot change the properties of a default messaging provider or a WebSphere MQ messaging provider.

To view this page, use the administrative console to complete one of the following steps:

- In the navigation pane, click **Resources** → **JMS** → **JMS providers**. This displays a list of JMS providers in the content pane. For each JMS provider in the list, the entry indicates the *scope* level at which JMS resource definitions are visible to applications. You can create the same type of JMS provider at different Scope settings, to offer JMS resources at different levels of visibility to applications.
- If you want to manage JMS resources that are defined at a different scope setting, change the **Scope** setting to the required level.
- In the Providers column of the list displayed, click the name of a JMS provider.

If you want to browse or change JMS resources of the JMS provider, click the link for the type of resource under Additional Properties. For more information about the administrative console panels for the types of JMS resources, see the related topics.

For default messaging providers and WebSphere MQ messaging providers, only the scope, name, and description properties are displayed for information only. You cannot change these properties.

For another type of JMS provider that you have defined yourself, extra properties are displayed that you can change.

The default messaging provider is installed and runs as part of WebSphere Application Server, and is based on service integration technologies.

Scope:

The level to which this resource definition is visible.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes. For more information about the scope setting, see Scope settings.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Default

- Default messaging provider.

  For JMS resources to be provided by a service integration bus, as part of WebSphere Application Server.
- *My JMSprovider*

  For JMS resources to be provided by your own JMS provider; not the default messaging provider or WebSphere MQ. You assign the name, in this example “My JMSprovider”, when you define the JMS provider to WebSphere Application Server. You must have installed and configured your own JMS provider before applications can use the JMS resources.
- WebSphere MQ messaging provider

  For JMS resources to be provided by WebSphere MQ. You must have installed and configured WebSphere MQ before applications can use the JMS resources.
- V5 default messaging provider.

  For JMS resources to be provided by a WebSphere Application Server Version 5 node.

Description:

A description of the JMS provider, for administrative purposes within WebSphere Application Server.

Data type String

Classpath:

A list of paths or JAR file names which together form the location for the JMS provider classes. Each class path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ’;’ or ’: ’). Class paths can contain variable (symbolic) names to be substituted using a variable map. Check your driver installation notes for specific JAR file names that are required.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

Native library path:

An optional path to any native libraries (*.dll, *.so). Each native path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ';' or ': '). Native paths can contain variable (symbolic) names to be substituted using a variable map.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

External initial context factory:

The Java classname of the initial context factory for the JMS provider.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type String
Default Null

External provider URL:

The JMS provider URL for external JNDI lookups.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, an LDAP URL for a messaging provider has the form: `ldap://hostname.company.com/contextName`.

Data type String
Default Null

Version 5 JMS server collection:

On a WebSphere Application Server Version 5 node, a JMS server provides the functions of the JMS provider. Use this panel to list JMS servers on WebSphere Application Server Version 5 nodes within the administration domain, or to select a JMS server to view or change its configuration properties.

There can be at most one JMS server on each Version 5 node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

To view this page, use the administrative console to complete the following step:
1. In the navigation pane, select **Servers** → **Version 5 JMS Servers**.

To browse or change the properties of a JMS server, select its name in the list displayed.

To act on one or more of the JMS servers listed, click the check box next to the server name, then use the buttons provided.

Version 5 JMS server settings:

The JMS functions on a Version 5 node are served by a JMS server. Use this panel to view or change the configuration properties of the selected JMS server.

JMS servers are supported only to aid migration of WebSphere Application Server Version 5 nodes to WebSphere Application Server Version 6.

You can use this panel to configure a general set of JMS server properties, which add to the default values of properties configured automatically for the Version 5 default messaging provider.

Note: JMS servers make use of WebSphere MQ properties and, in general, the default values of those properties are adequate. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, properties for log file locations, file pages, and buffer pages. For more information about configuring WebSphere MQ properties, see the *WebSphere MQ System Administration* book, SC33-1873, which is available from the IBM Publications Center or from the WebSphere MQ collection kit, SK2T-0730.

Name:

The name by which the JMS server is known for administrative purposes within IBM WebSphere Application Server.

This name should not be changed.

Data type	String
Units	Not applicable
Default	WebSphere Internal JMS Server
Range	Not applicable

Description:

A description of the JMS server, for administrative purposes within IBM WebSphere Application Server.

This string should not be changed.

Data type	String
Default	WebSphere Internal JMS Server

Number of threads:

The number of concurrent threads to be used by the publish/subscribe matching engine

The number of concurrent threads should only be set to a small number.

Data type	Integer
Units	Threads
Default	1
Range	Greater than or equal to 1.

Queue Names:

The names of the queues hosted by this JMS server. Each queue name must be added on a separate line.

Each queue listed in this field must have a separate queue administrative object with the same administrative name. To make a queue available to applications, define a WebSphere queue and add its name to this field on the JMS Server panel for the host on which you want the queue to be hosted.

Data type String
Units Queue name
Range Each entry in this field is a queue name of up to 45 characters, which must match exactly (including use of upper- and lowercase characters) the WebSphere queue administrative object defined for the queue.

Initial State:

The state that you want the JMS server to have when it is next restarted.

Data type Enum
Default Started
Range
Started
The JMS server is started automatically.
Stopped
The JMS server is not started automatically. If any deployed enterprise applications are to use JMS server functions provided by the JMS server, the system administrator must start the JMS server manually or select the Started value of this property then restart the JMS server.

To restart a JMS server on a Version 5 node, stop then restart that JMS server.

Version 5 WebSphere queue connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server Version 5 applications.

A WebSphere queue connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server Version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Queue connection factories**. This displays a list of any existing JMS queue connection factories.
5. Click the name of the JMS queue connection factory that you want to work with.

A queue connection factory for the embedded WebSphere JMS provider has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the JMS connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum

Default Null

Range Pull-down list of Version 5 nodes in the WebSphere administrative domain.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere queue connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum

Default Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type Checkbox

Default Selected (enabled for XA coordination)

Range

Selected
The connection factory is enabled for XA-coordination of messages

Cleared
The connection factory is not enabled for XA coordination of messages

Recommended Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

WebSphere topic connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server Version 5 applications.

A WebSphere topic connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server Version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Topic connection factories**. This displays a list of any existing JMS topic connection factories.
5. Click the name of the JMS topic connection factory that you want to work with.

A JMS topic connection factory for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type	String
Default	Null

JNDI name:

The JNDI name that is used to bind the topic connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type	String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type	String
Default	Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type	String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type	Enum
Default	Null
Range	Pull-down list of nodes in the WebSphere administrative domain.

Port:

Which of the two ports that connections use to connect to the JMS server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to `Direct` cannot be used with message-driven beans.

Data type Enum
Units Not applicable
Default QUEUED
Range

QUEUED
The listener port used for full-function JMS-compliant, publish/subscribe support.

DIRECT
The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not

valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum
Default Null
Range
ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Clone Support:

Select this checkbox to enable clone support to allow the same durable subscription across topic clones.

Data type Enum
Default Cleared
Range
Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to the queue manager.

Data type String
Range A valid JMS client ID

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback)

instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type	Checkbox
Default	Selected (enabled for XA coordination)
Range	**Selected** The connection factory is enabled for XA-coordination of messages **Cleared** The connection factory is not enabled for XA coordination of messages
Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Version 5 WebSphere queue destination settings:

Use this panel to view or change the configuration properties of the selected JMS queue destination for point-to-point messaging by WebSphere Application Server Version 5 applications.

A queue destination is used to configure a JMS queue of the default messaging provider for use by WebSphere Application Server Version 5 applications. Connections to the queue are created by the associated V5 Default Messaging WebSphere queue connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.

2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Queues**. This displays a list of any existing JMS queue destinations.
5. Click the name of the JMS queue destination that you want to work with.

A JMS queue for use with the internal WebSphere JMS provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

To enable applications to use this queue, you must add the queue name to the Queue Names field on the panel for the JMS server that hosts the queue.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.

NON PERSISTENT
Messages on the destination are not persistent.

PERSISTENT
Messages on the destination are persistent. When a persistent message is put to a queue, all of the message data is written to the messaging log (under the *embedded_messaging_install*\log directory) to make recovery of the message possible.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Default APPLICATION DEFINED
Range

APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.

QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	0
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Default	APPLICATION DEFINED
Range	**APPLICATION DEFINED** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **UNLIMITED** Messages on this queue have no expiry timeout, so those messages never expire. **SPECIFIED** The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Version 5 WebSphere topic destination settings:

Use this panel to browse or change the configuration properties of the selected JMS topic destination for publish/subscribe messaging by WebSphere application server Version 5 applications.

A WebSphere topic destination is used to configure the properties of a JMS topic for the default messaging provider on a Version 5 node. Connections to the topic are created by the associated topic connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the

cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.

3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Topics**. This displays a list of any existing JMS topic destinations.
5. Click the name of the JMS topic destination that you want to work with.

A JMS topic destination for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Topic:

The name of the topic as defined to the JMS provider.

Data type String
Default Null
Range The topic value can be dot notation and include wildcard characters.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON-PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type	Integer
Units	Message priority level
Default	0
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type	Enum
Units	Not applicable
Default	APPLICATION DEFINED
Range	**APPLICATION DEFINED** The expiry timeout for messages on this queue is defined by the application that put them onto the queue. **UNLIMITED** Messages on this queue have no expiry timeout, so those messages never expire. **SPECIFIED** The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Configuring Version 5 default JMS resources

Use the following tasks to configure the JMS connection factories and destinations WebSphere Application Server Version 5 applications.

About this task

You only need to complete these tasks if you have WebSphere application server Version 5 applications that need to use JMS resources provided by the default messaging provider. Such JMS resources are maintained as *V5 Default Messaging* resources.

- Configuring a connection for Version 5 default messaging
- Configuring a Version 5 default JMS queue connection factory
- Configuring a Version 5 default JMS topic connection factory
- Configuring a Version 5 default JMS queue destination

- Configuring a Version 5 default JMS topic destination

Configuring a connection for Version 5 default messaging:

Use this task to configure a connection to Version 5 default messaging. This task is provided to help you manually migrate nodes from v5 to v6. If you use the supplied tools to migrate existing v5 nodes to v6, the jmsserver, appserver and port that you create with this task are defined automatically.

About this task

To configure a connection for Version 5 default messaging, use the administrative console to complete the following steps:

1. Create an application server with the name jmsserver (only one of these can exist on each node). In the navigation pane, expand **Servers** → **Application servers**.
2. On the new server, define a new JMSSERVER_QUEUED_ADDRESS port with the host set to the v6 server host, and the port set to the same as the SIB_MQ_ENDPOINT_ADDRESS of the server which is a member of your bus. The appserver called jmsserver does not actually have to be started; it is only used for looking up the port address.
3. Define a queue connection factory and queue at the Node or Cell scope. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
4. In the content pane, click the name of the V5 default messaging provider.
5. Define a queue destination on your bus with the same name as your queue defined under v5 default messaging.
6. Define an alias destination on your bus with the same name as your queue defined under v5 default messaging but with WQ_ appended to the front the name. For example, if your queue has the name MyV5Queue, your alias should have the name WQ_MyV5Queue.
7. Point the alias at your queue destination with the correct name. The migration process targets the queue with the WQ_ prefix; defining the alias to point to the real queue helps migration.
8. Define the WebSphere MQ Client Link on your messaging engine on the bus. Keep the WebSphere MQ channel name WAS.JMS.SVRCONN and set the queue manager name so that it contains your node name. For example, if your node name is MyNode, you would set the queue manager name to WAS_<MyNode>_jmsserver. Now set the queue manager name to the same; in this example it would be WAS_MyNode_jmsserver. The WebSphere MQ Client Link will show a status of inactive, this is normal.
9. Restart your server so that the new JNDI definitions bind correctly. Your v5 client should now be able to connect to v6 using the host and bootstrap address of the v6 system in the provider url component of the initial context. Your client should now also be able to send messages to the destination on the bus.
10. If you open the Client connections view and click the refresh icon, the hostname of the connecting system is visible whilst the client is connected. In the navigation pane, expand**Buses** → **[bus name]** → **Messaging engines** → **[messaging engine name]** → **WebSphere MQ client links** → **[client link name]** → **Runtime** → **Client connections view**.
11. Click **OK**.
12. Save any changes to the master configuration.
13. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 queue connection factory:

Use this task to browse or change the properties of a JMS queue connection factory for point-to-point messaging with the default messaging provider on a Version 5 node. This task contains an optional step for you to create a new JMS queue connection factory.

About this task

To configure a JMS queue connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a queue connection factory.
3. In the content pane, under Additional Properties, click **Queue connection factories** This displays any existing JMS queue connection factories for the Version 5 messaging provider in the content pane.
4. To browse or change an existing JMS queue connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the JMS queue connection factory into the name space.

   c. Click **Apply**. This defines the JMS queue connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
5. Optional: Change properties for the queue connection factory, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 JMS topic connection factory:

Use this task to browse or change a JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server Version 5 applications.

About this task

To configure a JMS topic connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a topic connection factory.
3. Optional: Change the **Scope** setting to the level at which the JMS topic connection factory is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topic connection factories** This displays any existing JMS topic connection factories for the Version 5 messaging provider in the content pane.
5. To browse or change an existing JMS topic connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server.

JNDI Name
The JNDI name that is used to bind the JMS topic connection factory into the name space.

c. Click **Apply**. This defines the JMS topic connection factory to WebSphere Application Server, and enables you to browse or change additional properties.

6. Optional: Change properties for the topic connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere queue destination:

Use this task to browse or change the properties of a JMS queue destination for point-to-point messaging by WebSphere Application Server Version 5 applications. This task contains an optional step for you to create a new topic destination.

About this task

To configure a JMS queue destination for use by WebSphere Application Server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Optional: Change the **Scope** setting to the level at which the JMS destination is visible to applications.
3. Select the Version 5 provider for which you want to configure a queue destination.
4. In the content pane, under Additional Properties, click **Queues** This displays any existing queue destinations for the Version 5 default messaging provider in the content pane.
5. To browse or change an existing JMS queue destination, click its name in the list. Otherwise, to create a new queue destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this queue destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the queue destination into the name space.

   c. Click **Apply**. This defines the queue destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the queue destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To make a queue destination available to applications, host the queue on a JMS server. To add a new queue to a JMS server or to change an existing queue on a JMS server, you define the administrative name of the queue to the JMS server.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere topic destination:

Use this task to browse or change the properties of a JMS topic destination for publish/subscribe messaging by WebSphere application server Version 5 applications.. This task contains an optional step for you to create a new topic destination.

About this task

To configure a JMS topic destination for use WebSphere Application Server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a topic destination.
3. Optional: Change the **Scope** setting to the level at which the JMS destination is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topics** This displays any existing JMS topic destinations for the Version 5 default messaging provider in the content pane.
5. To browse or change an existing JMS topic destination, click its name in the list. Otherwise, to create a new topic destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this topic destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the topic destination into the name space.

      Topic The name of the topic in the default messaging provider, to which messages are sent.
   c. Click **Apply**. This defines the topic destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the topic destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring authorization security for a Version 5 default messaging provider

Use this task to configure authorization security for the default messaging provider on a WebSphere Application Server Version 5 node.

About this task

To configure authorization security for the Version 5 default messaging provider complete the following steps.

Note: Security for the Version 5 default messaging provider is enabled when you enable WebSphere Application Server security on the Version 5 node. For more information about enabling security, see Enabling security.

1. Configure authorization settings to access JMS resources owned by the embedded messaging subsystem.

   Authorization to access JMS resources owned by the embedded messaging subsystem is controlled by settings in the *profile_root*\config\cells*your_cell_name*\integral-jms-authorizations.xml file. The settings grant or deny authenticated users access to messaging resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Pub, sub, and persist to all topics.

To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.

2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

```
<queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
</queue-admin-userids>
```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.

3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

   For example, consider the following queue-default-permissions element:

```
<queue-default-permissions>
  <permission>write</permission>
</queue-default-permissions>
```

   In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

```
<topic>
   <name></name>
   <public>
     <permission>+pub</permission>
   </public>
 </topic>
```

In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
    <userid>useridpub</userid>
    <permission>+pub</permission>
  </authorize>
</topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

Results

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

Authorization settings for Version 5 default JMS resources:

Use the integral-jms-authorisations.xml file to view or change the authorization settings for Java Message Service (JMS) resources owned by the default messaging provider on WebSphere Application Server Version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*`\config\cells\`*your_cell_name*`\integral-jms-authorisations.xml` file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
```

```
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update: Controls whether or not the JMS Server checks dynamically for updates to this file.
true (Default) Enables dynamic update support.
false Disables dynamic update checking and improves authorization performance.

queue-admin-userids: This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:
<userid>*adminid***</userid>**

Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions: This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:
<permission>*read-write***</permission>**

Where *read-write* is one of the following keywords:
read By default, userids have read access to Version 5 default queue destinations.

write By default, userids have write access to Version 5 default queue destinations.

queue: This element contains the following authorization settings for a single queue destination:

name The name of the queue.

public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

topic: This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission

+sub Grant subscribe permission

+persist

Grant persist permission

-pub Deny publish permission
-sub Deny subscribe permission
-persist
Deny persist permission

Administering listener ports and activation specifications for message-driven beans

Use these tasks to manage the listener ports and activation specifications used by message-driven beans (MDBs). If the JMS provider is implemented as a J2EE Connector Architecture (JCA) resource adapter use an activation specification, otherwise use a listener port. These tasks are supplementary to the tasks of administering resource adapters, and JMS provider resources.

About this task

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

Use the WebSphere Application Server administrative console to configure the following resources for message-driven beans:

- J2C activation specifications for JCA 1.5-compliant message-driven beans. Activation specifications must be provided when the application's resources are configured using the default messaging provider or any generic J2C Resource Adapter that supports inbound messaging.
- The message listener service, listener ports, and listeners for EJB 2.0 message-driven beans deployed against listener ports. Listener ports must be provided when using the following JMS providers: V5 Default Messaging, or Generic.

Listener port or activation specification?

WebSphere Application Server Version 6 or later supports the enhancements made to the EJB 2.1 and Java EE specifications that allow any resource that has a JCA 1.5 adapter to trigger an MDB (formerly only JMS resources could trigger MDBs). WebSphere Application Server Version 5 and EJB 2.0 used the listener port which specified a JMS connection factory and a destination. This allowed a pool of MDB instances to connect to the messaging system and listen to the designated destination. EJB 2.1 specifies an additional requirement of an MDB working with a JCA adapter using an activation specification.

If you are developing WebSphere Application Server Version 6 or later applications, you should consider the following questions:

- Should I use a listener port or an activation specification?
- Should I convert my existing listener ports to activation specifications?
- Will listener ports eventually not be supported anymore?

Here are some guidelines to help you decide:

- If you are using J2EE 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used, so you do not have to make a decision. WebSphere Application Server Version 4 uses message beans, but these are not MDBs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server Version 5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server Version 5 uses listener ports to associate MDB classes with their JMS destinations.
- If you are using Java EE and EJB 2.1 with WebSphere Application Server Version 6 or later and not using JMS, you must use activation specifications. A connector MDB uses JCA to access its resources, so the connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.

- If you are using Java EE and EJB 2.1 with WebSphere Application Server Version 6 or later, the decision depends on whether your JMS provider API is implemented with JCA. In Java EE, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your MDB is a JMS MDB that is implemented as a connector MDB, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

To summarize: in WebSphere Application Server Version 6 or later, the decision on whether or not to use a listener port or an activation specification depends on the following scenarios:

- whether you upgrade your EJB 2.0 MDBs to EJB 2.1
- whether you want your EJB 2.1 MDB to use a JCA adapter
- whether you access your JMS provider via a JCA adapter.

If you have JMS API implementations that do not use JCA, WebSphere Application Server requires listener ports; if all JMS API implementations use JCA, listener ports are no longer required.

You can update the configuration data at any time, but some updates only take effect when the appropriate server is next started.

For additional information about administering support for message-driven beans, see the following topics:

- Configuring a JMS activation specification for MDBs used by the default messaging provider
- “Configuring a J2C activation specification”
- “Configuring a J2C administered object” on page 1732
- Configuring message listener resources for EJB 2.0 message-driven beans

Configuring a J2C activation specification

Use this task to configure a J2EE Connector (J2C) activation specification used to deploy message-driven beans with an external resource adapter.

About this task

Use this task if you want to use a message-driven bean as a listener on a Java Connector Architecture (JCA) 1.5 resource adapter other than the default messaging JMS provider.

You can create or modify a J2C activation specification under an installed resource adapter at the cell, node, or server scope. You can select the message listener type from those provided by the given resource adapter.

Configuring a J2C activation specification offers two distinct advantages:

- The activation specification configuration information can be shared among multiple message-driven beans across multiple applications.
- Updates to the configuration properties can be made without the need to redeploy the application.

The following guidelines show which scenarios use activation specifications or listener ports:

- If you are using J2EE 1.2 and Enterprise JavaBeans (EJB) 1.1 with WebSphere Application Server Version 4, message-driven beans are not used so you do not need listener ports or activation specifications. WebSphere Application Server Version 4 uses message beans, but these are not message-driven beans or enterprise beans.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server Version 5, you must use listener ports. The message-driven beans are JMS message-driven beans that implement MessageListener, and there is no JCA support. WebSphere Application Server Version 5 uses listener ports to associate message-driven bean classes with their JMS destinations.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server Version 6, you must use activation specifications. A connector message-driven bean uses JCA to access its resources, so the

connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of message-driven beans from EJB 2.0 to EJB 2.1.

- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server Version 6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your message-driven bean is a JMS message-driven bean that is implemented as a connector message-driven bean, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 message-driven bean in the same way as you would configure an EJB 2.0 message-driven bean, which in WebSphere Application Server is to use a listener port.

To configure a J2C activation specification for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new activation specification.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. Optional: Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Activation specifications heading, click **J2C Activation Specifications**. This lists any existing J2C activation specifications for the external resource adapter in the content pane.
4. Display the properties of the J2C activation specification. If you want to display an existing J2C activation specification, click one of the names listed.

   Alternatively, if you want to create a new J2C activation specification, click **New**, then specify the following required properties:

   Name Type the name by which the activation specification is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

   Message listener type
   Select the message listener type that this activation specification instance should support. This list is based on the deployment descriptor of the external resource adapter.

   Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new activation specification, you may need to click **Apply** before this custom property selection is available.
5. Specify properties for the activation specification, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Activation Specifications collection:

This page contains a list of J2C activation specifications for a resource adapter configuration and is used to create new J2C activation specifications, to select J2C activation specifications for configuration changes, or to delete J2C activation specifications.

Activation specification definitions and classes are provided by a resource adapter when it is installed. Using this information, the administrator can create and configure J2C activation specifications with JNDI names that are then available for applications to use. The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

The following guidelines show which scenarios use activation specifications or listener ports:

- If you are using Java 2 Platform, Enterprise Edition (J2EE) 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used so you do not need listener ports or activation specifications. WebSphere Application Server v4 uses message beans, but these are not MDbs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server v5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server v5 uses listener ;ports to associate MDB classes with their JMS destinations.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, you must use activation specifications. A connector MDB uses JCA to access its resources, so the connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your MDB is a JMS MDB that is implemented as a connector MDB, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C activation specifications**.
- **Resources** → **Resource Adapters** → **J2C activation specifications**.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Message Listener Type:

The Message Listener Type that is used by this activation specification.

The list of available classes is provided by the resource adapter.

Data type String

J2C Activation Specifications settings:

Use this page to specify the settings for a J2C activation specification.

The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C activation specifications** → ***activation_specification***.
- **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification***.

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification. Name is required

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Authentication alias:

This optional field is used to bind the J2C activation specification to an authentication alias (configured through the security JAAS screens).

This alias is used to access a user name and password that are set on the configured J2C activation specification. This field is only meaningful if the J2C activation specification you are configuring has a UserName and Password field.

If you have defined security domains in the application server, you can click **Browse...** to select a J2C authentication alias for the resource that you are configuring. Security domains allow you to isolate J2C authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Text

Message Listener Type:

The Message Listener Type used by this activation specification.

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. After you create the activation specification, the field is a read only text field.

Data type Drop-down list or text

Destination JNDIName:

The destination JNDIName field only appears when a message of type javax.jms.Destination with name *Destination* is received.

Configuring a J2C administered object

Use this task to configure a J2C administered object used to configure objects with an external resource adapter.

About this task

To configure a J2C administered object for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new administered object.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. Optional: Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Additional Properties heading, click **J2C Administered Objects**. This lists any existing J2C administered objects for the external resource adapter in the content pane.
4. Display the properties of the J2C administered object. If you want to display an existing J2C administered object, click one of the names listed.

Alternatively, if you want to create a new J2C administered object, click **New**, then specify the following required properties:

Name Type the name by which the J2C administered object is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

Administered object class
Select the administered object class that this instance should support. This list is based on the deployment descriptor of the external resource adapter.

Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new administered object, you may need to click **Apply** before this custom property selection is available.

5. Specify properties for the administered object, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Administered Objects collection:

Use this page to specify administered object settings for a Resource Adapter.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C administered objects**
- **Resources** → **Resource Adapters** → **Resource adapters** → **J2C administered objects**

Name:

Specifies display name assigned to this administered object.

Data type String

JNDI Name:

Specifies the JNDI name of the administered object.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Description:

Specifies a description for the administered object.

Data type String

Administered object class:

Specifies the Administered Object class that is associated with this J2C administered object. This class must be one that is provided by the resource adapter.

Data type String

J2C Administered Object settings:

Use this page to specify the settings for an administered object.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C administered objects** → ***J2C_administered_object***
- **Resources** → **Resource Adapters** → **Resource adapters** → **J2C administered objects** → ***J2C_administered_object***

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Data type String

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Data type String

Name:

Specifies the name of the J2C administered object instance.

A string with no spaces meant to be a meaningful text identifier for the administered object. This name is required.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name that this administered object is bound under.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Description:

Specifies a text description of the J2C administered object instance.

Data type String

Administered object class:

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. You can only select classes from this list.

After you create the administered object, you cannot modify the administered object class; it is read only.

Data type Class name

Configuring message listener resources for message-driven beans

Use the following tasks to configure resources needed by the message listener service to support message-driven beans for use with a JMS provider that does not have a J2EE Connector Architecture (JCA) 1.5 resource adapter.

About this task

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, such as the default messaging provider that is part of WebSphere Application Server Version 6. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA resources to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server Version 5).

Here are some guidelines on which scenarios use listener ports or activation specifications:

- If you are using J2EE 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used, so you do not used listener ports or activation specifications because WebSphere Application Server v4 uses message beans, but these are not MDbs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server v5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server v5 uses listener ports to associate MDB classes with their JMS destinations.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can be implemented with the JCA 1.5 API.

– If your JMS provider API is implemented with JCA, your MDB is a JMS MDB that is implemented as a connector MDB. A connector MDB uses JCA to access its resources, and so the connector must be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.
– If your JMS provider API is not implemented with JCA, you have the same JMS situation as for Java EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

If you want to deploy an enterprise application to use JMS message-driven beans with a JMS provider that does not have a JCA 1.5 resource adapter, refer to the following subtopics:

Configuring the message listener service:

Use this task to configure the properties of the message listener service for an application server, to support message-driven beans deployed against listener ports.

About this task

If the JMS provider does not have a J2EE Connector Architecture (JCA) 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server Version 5).

If you want to deploy an enterprise application to use message-driven beans with listener ports, you can use this task to browse or change the configuration of the message listener service for an application server.

To configure the message listener service for an application server, use the administrative console to complete the following steps:

1. Display the listener service settings page:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
2. Optional: Browse or change the value of properties for the message-driven bean thread pool.
   a. Click **Thread Pool**
   b. Change the following properties, to suit your needs:

      Minimum size
      The minimum number of threads to allow in the pool.

      Maximum size
      The maximum number of threads to allow in the pool.

      Thread inactivity timeout
      The number of milliseconds of inactivity that should elapse before a thread is reclaimed. A value of 0 indicates not to wait and a negative value (less than 0) means to wait forever.

      Note: The administrative console does not allow you to set the inactivity timeout to a negative number. To do this you must modify the value directly in the config.xml file.

Allow thread allocation beyond maximum thread size
Select this check box to enable the number of threads to increase beyond the maximum size configured for the thread pool.

c. Click **OK**.

3. Optional: Specify any of the following optional properties that you need, as **Custom properties** of the message listener service:

   NON.ASF.RECEIVE.TIMEOUT, MQJMS.POOLING.TIMEOUT, MQJMS.POOLING.THRESHOLD, MAX.RECOVERY.RETRIES, and RECOVERY.RETRY.INTERVAL.

   For more information about these custom properties, see Custom Properties.

   To browse or change the properties, complete the following steps:

   a. Click **Custom properties**
   b. For each custom property, specify a value to suit your needs.

      If you have not specified a property before:

      1) Click **New**.
      2) Type the name of the property.
      3) Type the value of the property.
      4) Click **OK**.

4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the Application Server.

Message listener service:

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service**

Custom Properties:

An optional set of name and value pairs for custom properties of the message listener service.

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL
- “DYNAMIC.CONFIGURATION.ENABLED” on page 1743

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use this panel to add new listener ports or to change the properties of existing listener ports.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

To view this administrative console panel, click **Servers** → **Application Servers** → *application_server* → **[Messaging] Message Listener Service** → **Listener Ports**

To manage a listener port, enable the **Select** check box beside the listener port name in the list and click a button:

Button	Resulting action
Convert to activation specification	Opens a wizard that helps you convert the selected listener port to an activation specification.
New	Accesses the panel to configure a new listener port.
Delete	Deletes the selected listener port or ports.
Start	Starts the selected listener port or ports.
Stop	Stops the selected listener port or ports.

For more information about asynchronous messaging, see “Asynchronous messaging in WebSphere Application Server using JMS” on page 266.

Listener port settings:

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Use this panel to view or change the configuration properties of the selected listener port.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*

Name:

The name by which the listener port is known for administrative purposes.

Data type String
Default Null

Initial state:

The state that you want the listener port to have when the application server is next restarted

Data type Enum
Units Not applicable
Default Started

Range

Started
When the application server is next started, the listener port is started automatically.

Stopped
When the application server is next started, the listener port is not started automatically. If message-driven beans are to use this listener port on the application server, the system administrator must start the port manually or select the Started value of this property then restart the application server.

Description:

A description of the listener port, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Connection factory JNDI name:

The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`.

Data type String
Default Null

Destination JNDI name:

The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

You cannot use a temporary destination for late responses.

Data type String
Default Null

Maximum sessions:

Specifies the maximum number of concurrent sessions that a listener can have with the JMS server to process messages.

Each session corresponds to a separate listener thread and therefore controls the number of concurrently processed messages. Adjust this parameter when the server does not fully use the available capacity of the machine and if you do not need to process messages in a specific message order.

Data type Integer
Units Sessions
Default 1
Range 1 through 2147483647

Recommended

- If you want to process messages in a strict message order, set the value to 1, so only one thread is ever processing messages.
- If you want to process multiple messages simultaneously (known as “message concurrency”), set this property to a value greater than 1. Keep this value as low as possible to prevent overloading client applications. A good starting point for a 100% JMS workload with short transaction times is 2 to 4 sessions per processor. If longer running transactions exist, you may need more sessions, which should be determined by experimentation.

Maximum retries:

The maximum number of times that the listener tries to deliver a message to a message-driven bean instance before the listener is stopped, in the range 0 through 2147483647.

Note: A WebSphere MQ queue has a similar property called the **BackoutThreshold** property. If your listener port is reading from a WebSphere MQ queue, then the retry limit and the behavior when the limit is reached is determined by whichever of these two properties is set to the lower limit:

- If you exceed the WebSphere MQ queue **BackoutThreshold** limit, the message that cannot be delivered is moved to somewhere else by WebSphere MQ (for example, to the WebSphere MQ backout requeue queue or the WebSphere MQ dead letter queue) and the listener port services the next message on the queue. In this case, WebSphere Application Server might not know that the message has not been delivered successfully.
- If you exceed the listener port **maximum retries** limit, the listener port stops. You then manually intervene to investigate the problem, possibly to remove the message from the WebSphere MQ queue then restart the listener port.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

Maximum messages:

The maximum number of messages that the listener can process in one transaction.

If the queue is empty, the listener processes each message when it arrives. Each message is processed within a separate transaction.

For the WebSphere V5 default messaging provider or WebSphere MQ as the JMS provider, if messages start accumulating on the queue then the listener can start processing messages in batches. For third-party messaging providers, this property value is passed to the JMS provider but the effect depends on the JMS provider.

Data type	Integer
Units	Number of messages
Default	1
Range	1 through 2147483647

Recommended	For the WebSphere default messaging providers or WebSphere MQ as the JMS provider, if you want to process multiple messages in a single transaction, then set this value to more than 1. If messages start accumulating on the queue, then a value greater than 1 enables multiple messages to be batch-processed into a single transaction, and eliminates much of the overhead of transactions on JMS messages. **Note:** • If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing. • Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch. • Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 can cause the transaction to time out. If an XA transaction does time out routinely because processing multiple messages exceeds the transaction timeout, reduce this property to 1 (to limit processing to one message per transaction) or increase your transaction timeout.

Message listener service custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the message listener service.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Custom Properties**

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL
- DYNAMIC.CONFIGURATION.ENABLED

NON.ASF.RECEIVE.TIMEOUT:

The timeout in milliseconds for synchronous message receives performed by message-driven bean listener sessions in the non-ASF mode of operation.

You should set this property to a non-zero value only if you want to enable the non-ASF mode of operation for all message-driven bean listeners on the application server.

The message listener service has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities (non-ASF).

- The ASF mode is meant to provide concurrency and transactional support for applications. For publish/subscribe message-drive beans, the ASF mode provides better throughput and concurrency, because in the non-ASF mode the listener is single-threaded.

- The non-ASF mode is mainly for use with third-party messaging providers that do not support JMS ASF, which is an optional extension to the JMS specification. The non-ASF mode is also transactional but, because the path length is shorter than the ASF mode, usually provides improved performance.

  Use non-ASF if:
  - Your third-party messaging provider does not provide JMS ASF support
  - You are using message-driven beans with WebSphere topic connections with the DIRECT port, because the embedded publish/subscribe broker using that port does not support XA transactions or JMS ASF.
  - Message order is a strict requirement

Data type Integer
Units Milliseconds
Default ASF mode (custom property not created)
Range 0 or greater milliseconds

0 non-ASF mode is disabled

1 or more
The timeout in milliseconds for non-ASF message-driven bean listener synchronous session receives

Recommended If a transaction timeout occurs, the message must recycle causing extra work. If you want to use the non-ASF mode, set this property to lower than the transaction timeout, but leave spare at least the maximum duration of your message-driven bean's onMessage() method. For example, if your message-driven bean's onMessage() method typically takes a maximum of 10 seconds, and the transaction timeout is set to 120 seconds, you might set the NON.ASF.RECEIVE.TIMEOUT property to no more than 110000 (110000 milliseconds, that is 110 seconds).

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type Integer
Units Milliseconds
Default 5 minutes
Range

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type Integer
Units Number of connections
Default 10
Range

MAX.RECOVERY.RETRIES:

The maximum number of times that a listener port managed by this service tries to recover from a failure before giving up and stopping. When stopped the associated listener port is changed to the stop state. The interval between retry attempts is defined by the RECOVERY.RETRY.INTERVAL custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Retry attempts
Default	5
Range	0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by a listener port to recover from a failure. The maximum number of retry attempts is defined by the MAX.RECOVERY.RETRIES custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Seconds
Default	60
Range	1 through 2147483647

DYNAMIC.CONFIGURATION.ENABLED:

This property controls whether the application server on which a listener port is created requires to be restarted. Set this property to true to enable dynamic configuration.

Data type	Boolean
Default	False (not selected)

Creating a new listener port:

Use this task to create a new listener port for the message listener service, so that message-driven beans can be associated with the port to retrieve messages.

About this task

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server Version 5), you are recommended to deploy such beans as J2EE Connector Architecture (JCA) 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you want to deploy an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to create a new listener port for a message-driven bean to retrieve messages from.

To create a new listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Click **New**.
3. Specify the following required properties:

   Name The name by which the listener port is known for administrative purposes.

   Connection factory JNDI name
   The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`

   Destination JNDI name
   The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.
4. Optional: Change other properties for the listener port, according to your needs.
5. Click **OK**.
6. Save your changes to the master configuration.
7. To have the changed configuration take effect, stop then restart the application server.

Results

If enabled, the listener port is started automatically when a message-driven bean associated with that port is installed.

Configuring a listener port:

Use this task to browse or change the properties of an existing listener port, used by message-driven beans associated with the port to retrieve messages.

About this task

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server Version 5), you are recommended to deploy such beans as J2EE Connector Architecture (JCA) 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you have deployed an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to browse or change the configuration of a listener port that a message-driven bean retrieves messages from.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

To configure the properties of a listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.

b. In the content pane, click the name of the application server.
c. Under Communications, click **Messaging** → **Message Listener Service**.
d. Click **Listener Ports**.

2. Click the name of the listener port that you want to work with. This displays the properties of the listener port in the content pane.
3. Optional: Change properties for the listener port, according to your needs.
4. Click **OK**.
5. Save any changes to the master configuration.
6. To have a changed configuration take effect, stop then restart the application server.

Deleting a listener port:

Use this task to delete a listener port from the message listener service, to prevent message-driven beans associated with the port from retrieving messages.

About this task

To delete a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. Under Communications, click **Messaging** → **Message Listener Service** This displays the Message Listener Service properties in the content pane.
4. In the content pane, click **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the check box for the listener port that you want to delete.
6. Click **Delete**. This action stops the port (needed to allow the port to be deleted) then deletes the port.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Administering listener ports:

Use the following tasks to administer listener ports, which each define the association between a connection factory, a destination, and a message-driven bean.

About this task

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

You can use the WebSphere Application Server administrative console to administer listener ports, as described in the following tasks.

Note: If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. You do not normally need to start or stop a listener port manually.

- Adding a new listener port Use this task to create a new listener port, to specify a new association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.
- Configuring a listener port Use this task to browse or change the configuration properties of a listener port.

- Starting a listener port Use this task to start a listener port manually.
- Stopping a listener port Use this task to stop a listener port manually.

Starting a listener port:

Use this task to start a listener port on an application server, to enable the listeners for message-driven beans associated with the port to retrieve messages.

About this task

A listener is active, that is able to receive messages from a destination, if the deployed message-driven bean, listener port, and message listener service are all started. Although you can start these components in any order, they must all be in a started state before the listener can retrieve messages.

If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. However, you can start a listener port manually, as described in this topic.

When a listener port is started, the listener manager tries to start the listeners for each message-driven bean associated with the port. If a message-driven bean is stopped, the port is started but the listener is not started, and remains stopped. If you start a message-driven bean, the related listener is started.

To start a listener port on an application server, use the administrative console to complete the following steps:

1. If you want the listener for a deployed message-driven bean to be able to receive messages at the port, check that the message-driven bean has been started.
2. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
3. Select the check box for the listener port that you want to start.
4. Click **Start**.
5. Save your changes to the master configuration.

Stopping a listener port:

Use this task to stop a listener port on an application server, to prevent the listeners for message-driven beans associated with the port from retrieving messages.

About this task

When you stop a listener port as described in this topic, the listener manager stops the listeners for all message-driven beans associated with the port.

To stop a listener port on an application server, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Select the check box for the listener port that you want to stop.

3. Click **Stop**.
4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the application server.

Message-driven beans - listener port components:

The WebSphere Application Server support for message-driven beans deployed against listener ports is based on JMS message listeners and the message listener service, and builds on the base support for JMS.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provided, refer to related tasks.

The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

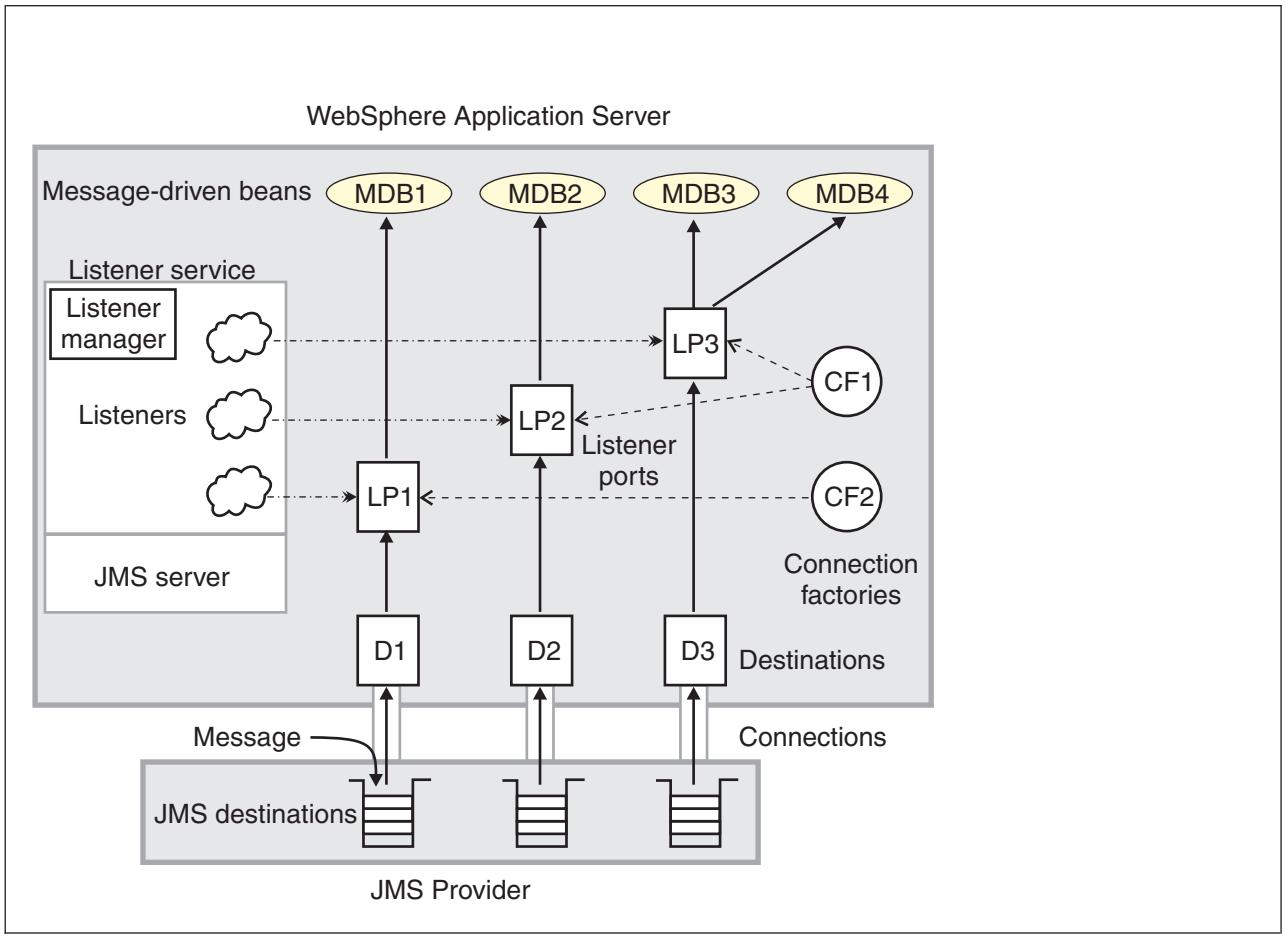


Figure 20. The main components for message-driven beans. This figure shows the main components of WebSphere support for message-driven beans, from JMS provider through a connection to a destination, listener port, then deployed message-driven bean that processes the message retrieved from the destination. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. The other main components are the message listener service, which comprises a listener for each listener port, all controlled by the same listener manager. For more information, see the text that accompanies this figure.

The *message listener service* is an extension to the JMS functions of the JMS provider and provides a *listener manager*, which controls and monitors one or more JMS *listeners*.

Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A *connection factory* is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A *listener port* defines the association between a connection factory, a destination, and a deployed *message-driven bean*. Listener ports are used to simplify the administration of the associations between these resources.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the listener manager based on the configuration data. The listener manager creates a dynamic session thread pool for use by listeners, creates and starts listeners, and during server termination controls the cleanup of listener message service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:
- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

Important file for message-driven beans

The *server_name*-durableSubscriptions.ser file in the WAS_HOME/temp directory is important for the operation of the WebSphere Application Server messaging service, so should not be deleted.

If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following file:

server_name-**durableSubscriptions.ser**

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

Choosing messaging providers for a mixed environment

If your existing or planned messaging environment involves both WebSphere MQ and WebSphere Application Server systems, choose between the default messaging provider, the WebSphere MQ messaging provider, or a mixture of the two, by considering your messaging requirements, your business environment, and the needs of each messaging application.

About this task

For messaging between application servers, with interaction with a WebSphere MQ system, you can use the default messaging provider or the WebSphere MQ provider. Neither provider is necessarily better than the other. The choice of providers depends primarily on factors relating to your business environment and planned changes to that environment, and also on what each JMS application needs to do. Moreover, these two types of messaging providers are not mutually exclusive:
- You can configure both types of providers within one cell.
- Different applications can use the same, or different, providers.

Factors relating to your business environment include the following:

- Messaging requirements
- Existing skill set
- Existing messaging infrastructure
- Planned changes to that infrastructure

Configuring and managing your messaging infrastructure is simpler if you use just one provider. If your messaging is primarily in WebSphere MQ, you should probably choose the WebSphere MQ messaging provider. Similarly, if your messaging is primarily in WebSphere Application Server, you should probably choose the default messaging provider.

If your business environment does not clearly indicate that you should use just one provider, then you should consider using a mixture of the two, and choosing the most appropriate messaging provider for each application. A useful way of doing this is to identify the types of destinations (service integration bus, or WebSphere MQ queue or topic) that the application is using. If the application uses only bus destinations, the natural choice is to use the default messaging provider (solution ″DMP″). If the application needs to communicate with one or more WebSphere MQ destinations, you can choose any of the following solutions depending upon your business environment, usage scenarios, and system topologies:

- Use the WebSphere MQ messaging provider (solution ″MQP″).
- Use the default messaging provider to integrate a WebSphere MQ server (a WebSphere MQ queue manager or queue-sharing group) as a bus member (solution ″DMP interop bus member″).
- Use the default messaging provider to integrate a WebSphere MQ network as a foreign bus, using WebSphere MQ links (solution ″DMP interop, foreign bus″).

For more information about these solutions, see Overview of interoperation with WebSphere MQ.

To help you choose between these solutions, several of the following steps contain tables in which each row represents a business or system requirement, and asterisks (*) indicate the solutions that are likely to be most effective for meeting the requirement. These tables are designed to provide general guidance, rather than to identify precisely a “right” solution. Most requirements have multiple possible solutions, and the absence of an asterisk does not necessarily mean that you cannot use that solution. To derive best guidance from using each of these tables:

- Focus on the rows that reflect your most important requirements.
- For all the rows that you consider, count the number of asterisks for each solution.

The solutions with the largest number of asterisks are likely to be the most effective.

1. If you have limited experience of WebSphere MQ or WebSphere Application Server, and are trying to decide which product best meets your messaging needs, see “Service integration and WebSphere MQ messaging - a comparison” on page 1688.

   Note: Whichever of these products you choose as the main focus for your messaging, you can still use either the default messaging provider or the WebSphere MQ provider for interoperation between the products.

2. Consider your business environment, to see if you can use just one provider.

   In deciding which provider to use, consider the following constraints:

   - The current and future messaging requirements
   - The existing messaging infrastructure
   - The skill set that you have in your organization

   If the majority of your messaging is today performed in WebSphere MQ, continue with that approach and configure WebSphere MQ as an external JMS provider (that is, use the WebSphere MQ messaging provider) in WebSphere Application Server. If the JMS requirements of your WebSphere Application Server applications are limited, it is debatable whether using a service integration bus for those applications gives sufficient benefit.

If you have messaging applications in WebSphere Application Server that have no requirement to interoperate with your WebSphere MQ network, use the default messaging provider (the service integration bus). If your WebSphere Application Server messaging requirements demand a tighter integration with WebSphere Application Server, the service integration bus provides the following benefits:

- Integrated administration
- WebSphere Application Server high availability capabilities
- WebSphere Application Server scalability

If you choose to use the default messaging provider to interoperate between service integration and WebSphere MQ, be aware that there is an added cost involved in converting messages between service integration format and WebSphere MQ format.

Also consider the following messaging scenarios:

- A large installed backbone of WebSphere MQ queue managers, perhaps with WebSphere Message Broker.

  If you want to use WebSphere Application Server to run a newly introduced messaging application, you can deploy a WebSphere Application Server (JMS) messaging application that will exchange messages with an existing application that uses a WebSphere MQ queue or topic.
- A WebSphere Application Server installation, perhaps with existing Web and enterprise applications, but no WebSphere Application Server messaging application.

  If you have no existing messaging infrastructure, you can deploy a WebSphere Application Server (JMS) messaging application to exchange messages with an existing WebSphere Application Server messaging application that uses a service integration bus destination.
- An infrastructure that uses WebSphere Application Server to connect WebSphere Application Server messaging applications.

  Introduce WebSphere Application Server (JMS) messaging between a pair of WebSphere Application Server applications.
- An infrastructure that includes both WebSphere MQ and service integration buses. This could be the result of a merger, or because the message traffic tends to be from WebSphere Application Server to WebSphere Application Server, or from WebSphere MQ to WebSphere MQ, but not typically between WebSphere Application Server and MQ.

  Deploy a WebSphere Application Server (JMS) messaging application to exchange messages with an application that uses a WebSphere MQ queue or topic.
- A WebSphere Process Server or WebSphere Enterprise Service Bus infrastructure, which uses Service Component Architecture (SCA).

  You can choose either a WebSphere MQ or a service integration bus binding for your SCA components.

3. If your business environment does not clearly indicate that you should use just one messaging provider, use a mixture of the two and choose the most appropriate provider for each application, based upon the destination types that the application uses.

   The application might need to exchange messages with existing partner applications or services that use one or more known destinations of known type. Alternatively, the partner applications or services might not yet be deployed and the choice of destination type might still be open, in which case the solution architect needs to decide how best to connect the applications or services together.

   If the application uses multiple destinations, there are four possible outcomes:

   - The application uses only bus destinations.
   - The application uses only WebSphere MQ destinations.
   - The application uses a mixture of bus and WebSphere MQ destinations.
   - The destination types are not yet known.

Note: If there is no clear business or technical reason why the application uses WebSphere MQ destinations rather than bus destinations, and the partner application is also a WebSphere Application Server JMS application, consider migrating the existing destinations to service integration so that the application uses only bus destinations.

4. If the application uses only bus destinations, configure the application and its JMS resources to use the default messaging provider.
5. If the application uses only WebSphere MQ destinations (queues or topics), use the following checklist to determine which provider solution to use.

Table 30. Provider checklist for an application that uses only WebSphere MQ destinations.

Question:	MQP	DMP interop, bus member	DMP interop, foreign bus
Is performance critical? (If so, use WebSphere MQ directly, rather than perform message conversion.)	*		
Does the application need to send or receive large messages (that is, messages > 500k.)?	*		
Is location transparency desirable for simplifying programming and deployment of applications?		*	*
Does the application need to consume from a WebSphere MQ queue, the configuration of which is fixed? (That is, the queue cannot be moved to service integration and you do not want to deploy a push-style WebSphere MQ application to send messages to a bus destination.)	*	*	
Is the partner application a JMS application that will run outside WebSphere Application Server, as a bus or WebSphere MQ client? (Do not mix service integration and WebSphere MQ unless you have to do so; a pure WebSphere MQ or service integration solution is simpler and avoids the cost of converting messages between service integration and WebSphere MQ formats.)	*		
Is the partner application a non-JMS (non-WebSphere Application Server) application? (Wherever possible choose a pure WebSphere MQ or service integration solution. Use the MQI WebSphere MQ client, or the XMS WebSphere MQ client, or the XMS bus client depending on your API preference.)	*		
Do you prefer traffic passing between your WebSphere MQ network and WebSphere Application Server applications to be funneled into a single long-running connection?			*
Do you want to use the high availability features of WebSphere Application Server?			*
Is XA 2pc needed between the application and a WebSphere MQ queue sharing group?		*	*
Is XA 2pc needed between the application and a WebSphere MQ cluster?		*	

6. If the application uses a mixture of bus and WebSphere MQ destinations, for example consuming from service integration and sending to WebSphere MQ, then either of the default messaging provider interoperation models can support this by using a single connection factory or activation specification. Use the following checklist to help you decide between a bus member and a foreign bus solution.

Table 31. Provider checklist for an application that uses a mixture of bus and WebSphere MQ destinations.

Question:	DMP interop, bus member	DMP interop, foreign bus
Does the application need to consume from a WebSphere MQ shared queue?	*	

Table 31. Provider checklist for an application that uses a mixture of bus and WebSphere MQ destinations. (continued)

Question:	DMP interop, bus member	DMP interop, foreign bus
Is there a need to distribute work to a pool of WebSphere Application Server workers from a WebSphere MQ queue?	*	
Do you prefer traffic passing between your WebSphere MQ network and WebSphere Application Server applications to be funneled into a single long-running connection?		*
Do you need distributed WebSphere MQ in versions earlier than WebSphere Application Server Version 7 and WebSphere MQ Version 7?		*
Do you want store and forward capabilities to allow the application to continue to send messages when the WebSphere MQ queue manager is unavailable?		*
Do you prefer not to configure server connection channels? (This is because they open a port, which could be seen as a security risk.)		*
Do you prefer to define a server connection channel, rather than a pair of sender and receiver channels?	*	

7. If the destination types are not yet known, decide the relative priorities of the known concerns then use the following checklist to assess how well each of them is addressed by the possible provider solutions.

   The choice is really over what type of destinations this application should use. The destination types are not yet fixed, so any of the four solutions is possible, but as a general rule you should aim for solution “DMP” or “MQP”, because a pure WebSphere MQ or service integration solution is simpler and avoids the cost of converting messages between service integration and WebSphere MQ formats.

Table 32. Provider checklist for an application for which the destination types are not yet known.

Question:	DMP	MQP	DMP interop, bus member	DMP interop, foreign bus
Do you have an existing base of strong skills in managing WebSphere MQ?		*	*	*
Do you want management of all messaging to be handled by the WebSphere MQ team?		*		
Do you have administrators skilled in WebSphere Application Server but not in WebSphere MQ?	*			
Do you want a messaging product with a large installed base (including references) and a wide choice of ISV tools?		*		
Are you reluctant to buy a separately licensed product in addition to WebSphere Application Server?	*			
Are you reluctant to install and manage a separate product in addition to WebSphere Application Server?	*			
Are you already using WebSphere Message Broker? (If so, you need WebSphere MQ anyway).		*	*	*
Are you using the WebSphere Enterprise Service Bus to mediate messages from, or deliver them to, a WebSphere MQ queue? (For example, using WebSphere Business Integration adapters, or connecting to a service provider such as CICS.)		*		

Table 32. Provider checklist for an application for which the destination types are not yet known. (continued)

Question:	DMP	MQP	DMP interop, bus member	DMP interop, foreign bus
Does the application need to send or receive large messages (that is, messages > 500k.)?		*		
Is location transparency desirable for simplifying programming and deployment of applications?	*		*	*
Do the throughput requirements need multiple parallel channels or routes?		*	*	*
Does the application need to consume from a WebSphere MQ queue, the configuration of which is fixed? (that is, the queue cannot be moved to service integration and you don't want to deploy a push-style WebSphere MQ application to send messages to a bus destination.)	*	*	*	
Is the partner application a JMS application that will also run in WebSphere Application Server? (Service integration runs in the WebSphere Application Server application server. On distributed platforms that means it is in-process. On the z/OS platform it is in another region.)	*			
Is the partner application a JMS application that will run outside WebSphere Application Server, as a bus or WebSphere MQ client? (Do not mix service integration and WebSphere MQ unless you have to do so; a pure WebSphere MQ or service integration solution is simpler and avoids the cost of converting messages between service integration and WebSphere MQ formats.)	*	*		
Is the partner application a non-JMS (non-WebSphere Application Server) application? (Wherever possible choose a pure WebSphere MQ or service integration solution. Use the MQI WebSphere MQ client, or the XMS WebSphere MQ client, or the XMS bus client depending on your API preference.)	*	*		
Is maintenance of strict message order important?	*			
Does the application require the flexibility and convenience of a WebSphere MQ cluster? (WebSphere MQ clustering makes administration simpler and provides selective parallelism of clustered queues. That is, instances of a clustered queue can be created on any (but not necessarily all) queue managers in the WebSphere MQ cluster. Messages sent to the clustered queue can be addressed to a specific instance of the queue, or allowed to select an instance dynamically based on workload management statistics. WebSphere Application Server clustering provides some of this flexibility, but you cannot create partitions of a bus destination on a subset of the messaging engines in a cluster bus member.)		*	*	*
Does the application need the level of high availability provided by WebSphere MQ for z/OS shared queues?		*	*	*
Do you want to use the high availability or scalability features of WebSphere Application Server clustering?	*		*	*

Service integration and WebSphere MQ messaging - a comparison

If you are not already an established user of either WebSphere Application Server or WebSphere MQ, and you are considering whether the service integration platform or WebSphere MQ better meets your messaging needs, use this table to compare the main features of the two platforms.

Table 33. Comparison of service integration and WebSphere MQ main features

service integration (the default messaging provider for WebSphere Application Server)	**WebSphere MQ**
Service integration is closely integrated with WebSphere Application Server, and is a natural fit if you are using the Java Platform, Enterprise Edition (Java EE).	WebSphere MQ can connect to almost anything. It provides a very heterogeneous environment .
Service integration supports multiple languages through XMS clients, and multiple platforms.	WebSphere MQ supports multiple languages and multiple platforms.
Service integration is a single process, pure Java implementation.	WebSphere MQ has many Independent Software Vendor (ISV) tools.
Service integration provides strong performance for both persistent and non-persistent messages for JMS.	WebSphere MQ supports JMS and non-JMS messaging interfaces, and provides strong performance for non-JMS applications.
Service integration is designed for a maximum message size of about 40 megabytes on a 32-bit operating system (subject to heap usage).	WebSphere MQ supports large message sizes up to about 100 megabytes.
Service integration is tightly integrated with some Web services implementations.	WebSphere MQ is a natural fit if you are using WebSphere Message Brokers.
Service integration messaging is included in a single administrative model for WebSphere Application Server, WebSphere Enterprise Service Bus and WebSphere Process Server.	WebSphere MQ can integrate existing infrastructure and applications (for example CICS).
Service integration bus clustering is integrated with WebSphere Application Server clustering for high availability and scalability.	WebSphere MQ clustering provides selective parallelism of clustered queues.

Note: The messaging platform that you choose for a given task does not necessarily determine which JMS messaging provider you should use. For example:

- If you need to handle large messages, you probably need to use the WebSphere MQ messaging provider.
- If you are using WebSphere Message Brokers, you can use either the default messaging provider or the WebSphere MQ messaging provider.

Other ways of managing messaging

For messaging between application servers, most requirements are best met by either the default messaging provider or the WebSphere MQ messaging provider. However, you can instead use a third-party messaging provider (that is, use another company's product as the provider). For backwards compatibility with earlier releases, there is also support for the V5 default messaging provider.

Before you begin

If you are not sure which provider combination is best suited to your needs, see "Types of messaging providers" on page 1814.

About this task

Enterprise applications in WebSphere Application Server can use asynchronous messaging through services based on Java Message Service (JMS) messaging providers and their related messaging systems. These messaging providers conform to the JMS Version 1.1 specification.

The choice of provider depends on what your JMS application needs to do, and on other factors relating to your business environment and planned changes to that environment.

- Choose a third-party messaging provider.

  You can configure any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

  To administer a third-party messaging provider, you use the resource adaptor or client supplied by the third party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

  To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the Java EE Connector Architecture (JCA) Version 1.5 specification.

  To work with a third-party provider, see “Managing messaging with a third-party messaging provider” on page 1689.
- Choose the (deprecated) V5 default messaging provider.

  This deprecated provider is identical to the WebSphere Application Server Version 5 default provider. Only the name has changed. It provides backwards compatibility that enables WebSphere Application Server Version 6 or later applications to connect to WebSphere Application Server Version 5 resources in a mixed cell. It also allows WebSphere Application Server Version 5 applications to connect to WebSphere Application Server Version 6 or later resources in a mixed cell. To configure and manage messaging to interoperate with WebSphere Application Server Version 5, see “Maintaining Version 5 default messaging resources” on page 1699.

Managing messaging with a third-party messaging provider

Enabling your messaging applications to use JMS resources provided by a third-party messaging provider other than WebSphere MQ.

Before you begin

For messaging between application servers, perhaps with some interaction with a WebSphere MQ system, you can use the default messaging provider. To integrate WebSphere Application Server messaging into a predominately WebSphere MQ network, you can use the WebSphere MQ messaging provider. You can also use a third-party messaging provider as described in this topic. To choose the provider that is best suited to your needs, see “Choosing a messaging provider” on page 1676.

About this task

You can install a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider.

WebSphere Application Server applications can use the JMS 1.1 interfaces or JMS 1.0.2 interfaces to access JMS resources provided by a third-party messaging provider, and you can use the administrative console to administer the JMS connection factories and destinations for the provider.

To use a third-party messaging provider with WebSphere Application Server, complete one or more of the following steps:

- Define a third-party messaging provider.
- List third-party JMS messaging resources.
- Configure JMS resources for a third-party messaging provider.

Defining a third-party messaging provider

Use this task to define a third-party messaging provider to WebSphere Application Server, for use instead of the default messaging provider or WebSphere MQ messaging provider.

Before you begin

Before you configure a third-party messaging provider, you might want to check whether your requirement can be met by the default messaging provider or the WebSphere MQ messaging provider that are supplied with WebSphere Application Server. To choose the provider that is best suited to your needs, see “Choosing a messaging provider” on page 1676.

About this task

You can configure any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

To administer a third-party messaging provider, you use the resource adaptor or client supplied by the third-party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the J2EE Connector Architecture (JCA) Version 1.5 specification.

To define a new third-party messaging provider to WebSphere Application Server, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources** → **JMS** → **JMS Providers** . The existing messaging providers are displayed, including the default messaging provider and the WebSphere MQ messaging provider.
2. To define a new third-party messaging provider, click **New** in the content pane. Otherwise, to change the definition of an existing messaging provider, click the name of the provider.
3. Specify the following required properties. You can specify other properties, as described in a later step.

   Name The name by which this messaging provider is known for administrative purposes within IBM WebSphere Application Server.

   External initial context factory
   The Java classname of the initial context factory for the JMS provider.

   External provider URL
   The JMS provider URL for external JNDI lookups.
4. Optional: Click **Apply**. This enables you to specify additional properties.
5. Optional: Specify other properties for the messaging provider.

   Under Additional Properties, you can use the **Custom Properties** link to specify custom properties for your initial context factory, in the form of standard javax.naming properties.
6. Click **OK**.
7. Save the changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

What to do next

You can now configure JMS resources for your messaging provider, as described in “Configuring JMS resources for a third-party messaging provider” on page 1697.

Listing third-party JMS messaging resources

Use this task with the WebSphere Application Server administrative console to display administrative lists of JMS resources provided by a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider.

About this task

You can use the WebSphere Application Server administrative console to display lists of the following types of JMS resources provided by a 3rd party messaging provider. You can use the panels displayed to select JMS resources to administer, or to create or delete JMS resources (where appropriate).

To display administrative lists of JMS resources for a third-party messaging provider, complete the following steps:

1. Start the administrative console.
2. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
3. Click the name of the third-party messaging provider.
4. In the content pane, under Additional Resources, click the link for the type of JMS resource. For more information about the detailed settings displayed for each resource type, see the related reference topics.

JMS provider collection:

Use this panel to list JMS providers, or to select a JMS provider to view or change its configuration properties.

To view this administrative console page, click **Resources** → **JMS** → **JMS providers**

To view or change the properties of a JMS provider or its resources, select its name in the list displayed.

To define a new third-party messaging provider, click **New**.

To act on one or more of the JMS providers listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Name The name by which this JMS provider is known for administrative purposes.

Description

A description of this JMS provider for administrative purposes.

For related information about JMS messaging providers and asynchronous messaging, see

- "Types of messaging providers" on page 1814
- "Asynchronous messaging in WebSphere Application Server using JMS" on page 266

Third-party JMS connection factory collection:

The JMS connection factories configured for a third-party messaging provider for both point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS connection factories, or to select a connection factory to browse or change its configuration properties.

This panel shows a list of the JMS connection factories configured for a third-party messaging provider, with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS connection factory.
3. Under Additional Properties, click **Connection factories**.

To define a new JMS connection factory, click **New**.

To view or change the properties of a JMS connection factory, select its name in the list displayed.

To act on one or more of the JMS connection factories listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Third-party JMS connection factory settings:

Use this panel to browse or change the configuration properties of a JMS connection factory configured for use with a third-party messaging provider. These configuration properties control how connections are created to the JMS destinations on the provider.

A JMS connection factory is used to create connections to JMS destinations. The JMS connection factory is created by the associated JMS provider.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. If appropriate, in the content pane, change the scope of the third-party messaging provider.
3. In the content pane, click the name of the messaging provider that supports the JMS connection factory.
4. Under Additional Properties, click **Connection factories**.
5. Click the name of the JMS connection factory that you want to work with.

A JMS connection factory for a third-party messaging provider (that is, a provider other than the default, the V5 default or the WebSphere MQ messaging provider) has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the associated messaging provider.

Data type String

Type:

Whether this connection factory is for creating JMS queue destinations or JMS topic destinations.

Select one of the following options:

QUEUE
A JMS queue connection factory for point-to-point messaging.

TOPIC
A JMS topic connection factory for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the connection factory into the WebSphere Application Server name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the connection factory into the name space of the third-party messaging provider.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to a JMS provider for container-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field provides a list of the modules that have been configured on the **Global Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum

Default Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Custom properties:

An optional set of name and value pairs for custom properties passed to the messaging provider.

Third-party JMS destination collection:

The JMS destinations configured in an associated third-party messaging provider for point-to-point and publish/subscribe messaging. Use this panel to create or delete JMS destinations, or to select a JMS destination to browse or change its configuration properties.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging.

To define a new JMS destination, click **New**.

To view or change the properties of a JMS destination, select its name in the list displayed.

To act on one or more of the JMS destinations listed, click the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

Third-party JMS destination settings:

Use this panel to browse or change the configuration properties of the selected JMS destination for use with an associated third-party messaging provider.

A JMS destination is used to configure the properties of a JMS destination for the associated third-party messaging provider (that is, not the default messaging provider or the WebSphere MQ messaging provider). Connections to the JMS destination are created by the associated JMS connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. In the content pane, click the name of the third-party messaging provider that you want to support the JMS destination.
3. Under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging.
4. Click the name of the JMS destination that you want to work with.

A JMS destination for use with a third-party messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

Type:

Whether this JMS destination is a queue (for point-to-point) or topic (for publish/subscribe).

Select one of the following options:
Queue
A JMS queue destination for point-to-point messaging.
Topic A JMS topic destination for publish/subscribe messaging.

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String

Category:

A category used to classify or group this queue, for your WebSphere Application Server administrative records.

Data type String

External JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Configuring JMS resources for a third-party messaging provider

Use the following tasks to configure the JMS connection factories and destinations for a third-party messaging provider (that is, a messaging provider other than the default messaging provider or the WebSphere MQ messaging provider).

Before you begin

You only need to complete these tasks if your WebSphere Application Server environment uses a third-party messaging provider to support enterprise applications that use JMS. To enable use of a third-party messaging provider, you must have installed and configured the messaging provider, as described in Defining a third-party messaging provider.

About this task

To configure JMS resources for a third-party messaging provider, complete the following tasks:

- Configure a JMS connection factory for a third-party messaging provider.
- Configure a JMS destination for a third-party messaging provider.

Configuring a JMS connection factory for a third-party messaging provider:

Use this task to browse or change the properties of a JMS connection factory for use with a JMS messaging provider other than the default, V5 default or WebSphere MQ messaging providers.

About this task

To configure a JMS connection factory for use with a third-party messaging provider, use the administrative console to complete the following steps:

1. Display the third-party messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS Providers** .
2. Select the third-party provider for which you want to configure a connection factory.
3. Optional: Change the **Scope** setting to the level at which the connection factory is visible to applications.
4. In the content pane, under Additional Properties, click **Connection factories** This displays a table listing any existing JMS connection factories, with a summary of their properties.
5. To browse or change an existing JMS connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS connection factory is known for administrative purposes within IBM WebSphere Application Server.

      Type Select whether the connection factory is for JMS queues (QUEUE) or JMS topics (TOPIC).

      JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the WebSphere Application Server name space.

      External JNDI Name
      The JNDI name that is used to bind the JMS connection factory into the name space of the messaging provider.

   c. Click **Apply**. This defines the JMS connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the JMS connection factory, according to your needs.

7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS destination for a third-party messaging provider:

Use this task to browse or change the properties of a JMS destination for use with a third-party messaging provider (that is, a provider other than the default messaging provider or the WebSphere MQ messaging provider).

About this task

To configure a JMS destination for use with a third-party messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. Click the name of the third-party messaging provider.
3. In the content pane, under Additional Properties, click **Queues** for point-to-point messaging or **Topics** for publish/subscribe messaging. This displays a table listing any existing JMS destinations, with a summary of their properties.
4. To browse or change an existing JMS destination, click its name in the list. Otherwise, to create a new destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS destination is known for administrative purposes within WebSphere Application Server.

      Type Select whether the destination is for JMS queues (QUEUE) or JMS topics (TOPIC).

      JNDI Name
      The JNDI name that is used to bind the JMS destination into the WebSphere Application Server name space.

      External JNDI Name
      The JNDI name that is used to bind the JMS destination into the name space of the messaging provider.

   c. Click **Apply**. This defines the JMS destination to WebSphere Application Server, and enables you to browse or change additional properties.
5. Optional: Change properties for the JMS destination, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Maintaining Version 5 default messaging resources

This topic is the entry-point into a set of topics about maintaining messaging resources provided for WebSphere Application Server Version 5.1 applications by the default messaging provider.

About this task

JMS applications running on WebSphere Application Server Version 5.1 can use JMS resources provided by the default messaging provider in WebSphere Application Server Version 7. You can use the

WebSphere Application Server administrative console to manage the JMS connection factories and destinations for WebSphere Application Server Version 5.1 applications. Such JMS resources are maintained as *V5 Default Messaging* resources.

V5 Default Messaging provides a JMS transport to a messaging engine of a service integration bus that supports the default messaging provider in WebSphere Application Server Version 7. The messaging engine emulates the service of a JMS server running on WebSphere Application Server Version 5.1.

You can also use the administrative console to manage a JMS server on a Version 5.1 node.

- List Version 5 default messaging resources.
- Configure Version 5 default JMS resources.
- Configure authorization security for a Version 5 default messaging provider.

Listing Version 5 default messaging resources

Use the WebSphere Application Server administrative console to list JMS resources for the Version 5 default messaging provider, for administrative purposes.

About this task

You use the WebSphere Application Server administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:

- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue
- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:

- Provider-centric navigation. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS providers** is provider-centric.
- Resource-centric navigation lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → **[resource]**, where [resource] is one of the resource types listed above is resource-centric.

You can use either of these navigation pathways to locate resources of any type.

- Using provider-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
  3. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
  4. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.

5. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties

- Using resource-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.
  3. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties.

JMS provider settings:

Use this panel to view the configuration properties of a selected JMS provider. You cannot change the properties of a default messaging provider or a WebSphere MQ messaging provider.

To view this page, use the administrative console to complete one of the following steps:

- In the navigation pane, click **Resources** → **JMS** → **JMS providers**. This displays a list of JMS providers in the content pane. For each JMS provider in the list, the entry indicates the *scope* level at which JMS resource definitions are visible to applications. You can create the same type of JMS provider at different Scope settings, to offer JMS resources at different levels of visibility to applications.
- If you want to manage JMS resources that are defined at a different scope setting, change the **Scope** setting to the required level.
- In the Providers column of the list displayed, click the name of a JMS provider.

If you want to browse or change JMS resources of the JMS provider, click the link for the type of resource under Additional Properties. For more information about the administrative console panels for the types of JMS resources, see the related topics.

For default messaging providers and WebSphere MQ messaging providers, only the scope, name, and description properties are displayed for information only. You cannot change these properties.

For another type of JMS provider that you have defined yourself, extra properties are displayed that you can change.

The default messaging provider is installed and runs as part of WebSphere Application Server, and is based on service integration technologies.

Scope:

The level to which this resource definition is visible.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes. For more information about the scope setting, see Scope settings.

Name:

The name by which the JMS provider is known for administrative purposes.

Data type String

Default

- Default messaging provider.

  For JMS resources to be provided by a service integration bus, as part of WebSphere Application Server.
- *My JMSprovider*

  For JMS resources to be provided by your own JMS provider; not the default messaging provider or WebSphere MQ. You assign the name, in this example “My JMSprovider”, when you define the JMS provider to WebSphere Application Server. You must have installed and configured your own JMS provider before applications can use the JMS resources.
- WebSphere MQ messaging provider

  For JMS resources to be provided by WebSphere MQ. You must have installed and configured WebSphere MQ before applications can use the JMS resources.
- V5 default messaging provider.

  For JMS resources to be provided by a WebSphere Application Server Version 5 node.

Description:

A description of the JMS provider, for administrative purposes within WebSphere Application Server.

Data type String

Classpath:

A list of paths or JAR file names which together form the location for the JMS provider classes. Each class path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ’;’ or ’: ’). Class paths can contain variable (symbolic) names to be substituted using a variable map. Check your driver installation notes for specific JAR file names that are required.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

Native library path:

An optional path to any native libraries (*.dll, *.so). Each native path entry is on a separate line (separated by using the Enter key) and must not contain path separator characters (such as ’;’ or ’: ’). Native paths can contain variable (symbolic) names to be substituted using a variable map.

This property does not apply to default messaging providers or WebSphere MQ providers.

Data type String

External initial context factory:

The Java classname of the initial context factory for the JMS provider.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, for an LDAP service provider the value has the form: `com.sun.jndi.ldap.LdapCtxFactory`.

Data type	String
Default	Null

External provider URL:

The JMS provider URL for external JNDI lookups.

This property does not apply to default messaging providers or WebSphere MQ providers.

For example, an LDAP URL for a messaging provider has the form: `ldap://hostname.company.com/contextName`.

Data type	String
Default	Null

Version 5 JMS server collection:

On a WebSphere Application Server Version 5 node, a JMS server provides the functions of the JMS provider. Use this panel to list JMS servers on WebSphere Application Server Version 5 nodes within the administration domain, or to select a JMS server to view or change its configuration properties.

There can be at most one JMS server on each Version 5 node in the administration domain, and any application server within the domain can access JMS resources served by any JMS server on any node in the domain.

To view this page, use the administrative console to complete the following step:
1. In the navigation pane, select **Servers** → **Version 5 JMS Servers**.

To browse or change the properties of a JMS server, select its name in the list displayed.

To act on one or more of the JMS servers listed, click the check box next to the server name, then use the buttons provided.

Version 5 JMS server settings:

The JMS functions on a Version 5 node are served by a JMS server. Use this panel to view or change the configuration properties of the selected JMS server.

JMS servers are supported only to aid migration of WebSphere Application Server Version 5 nodes to WebSphere Application Server Version 6.

You can use this panel to configure a general set of JMS server properties, which add to the default values of properties configured automatically for the Version 5 default messaging provider.

Note: JMS servers make use of WebSphere MQ properties and, in general, the default values of those properties are adequate. However, if you are running high messaging loads, you may need to change some WebSphere MQ properties; for example, properties for log file locations, file pages, and buffer pages. For more information about configuring WebSphere MQ properties, see the *WebSphere MQ System Administration* book, SC33-1873, which is available from the IBM Publications Center or from the WebSphere MQ collection kit, SK2T-0730.

Name:

The name by which the JMS server is known for administrative purposes within IBM WebSphere Application Server.

This name should not be changed.

Data type	String
Units	Not applicable
Default	WebSphere Internal JMS Server
Range	Not applicable

Description:

A description of the JMS server, for administrative purposes within IBM WebSphere Application Server.

This string should not be changed.

Data type	String
Default	WebSphere Internal JMS Server

Number of threads:

The number of concurrent threads to be used by the publish/subscribe matching engine

The number of concurrent threads should only be set to a small number.

Data type	Integer
Units	Threads
Default	1
Range	Greater than or equal to 1.

Queue Names:

The names of the queues hosted by this JMS server. Each queue name must be added on a separate line.

Each queue listed in this field must have a separate queue administrative object with the same administrative name. To make a queue available to applications, define a WebSphere queue and add its name to this field on the JMS Server panel for the host on which you want the queue to be hosted.

Data type	String
Units	Queue name
Range	Each entry in this field is a queue name of up to 45 characters, which must match exactly (including use of upper- and lowercase characters) the WebSphere queue administrative object defined for the queue.

Initial State:

The state that you want the JMS server to have when it is next restarted.

Data type	Enum
Default	Started

Range

Started
The JMS server is started automatically.

Stopped
The JMS server is not started automatically. If any deployed enterprise applications are to use JMS server functions provided by the JMS server, the system administrator must start the JMS server manually or select the Started value of this property then restart the JMS server.

To restart a JMS server on a Version 5 node, stop then restart that JMS server.

Version 5 WebSphere queue connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS queue connection factory for point-to-point messaging for use by WebSphere Application Server Version 5 applications.

A WebSphere queue connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server Version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Queue connection factories**. This displays a list of any existing JMS queue connection factories.
5. Click the name of the JMS queue connection factory that you want to work with.

A queue connection factory for the embedded WebSphere JMS provider has the following properties:

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the JMS connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Default Null
Range Pull-down list of Version 5 nodes in the WebSphere administrative domain.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere queue connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

This field is deprecated in 6.0. The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type Enum

Default Null

Range

ClientContainer
The client container maps authentication aliases.

WSLogin
The WSLogin module maps authentication aliases.

DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type	Checkbox
Default	Selected (enabled for XA coordination)
Range	**Selected** The connection factory is enabled for XA-coordination of messages **Cleared** The connection factory is not enabled for XA coordination of messages
Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

WebSphere topic connection factory settings:

Use this panel to browse or change the configuration properties of the selected JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server Version 5 applications.

A WebSphere topic connection factory is used to create JMS connections to the default messaging provider for use by WebSphere Application Server Version 5 applications.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Topic connection factories**. This displays a list of any existing JMS topic connection factories.
5. Click the name of the JMS topic connection factory that you want to work with.

A JMS topic connection factory for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server. The name must be unique within the JMS connection factories across the WebSphere administrative domain.

Data type String
Default Null

JNDI name:

The JNDI name that is used to bind the topic connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of this topic connection factory for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic connection factory, for your IBM WebSphere Application Server administrative records.

Data type String

Node:

The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.

Data type Enum
Default Null
Range Pull-down list of nodes in the WebSphere administrative domain.

Port:

Which of the two ports that connections use to connect to the JMS server. The QUEUED port is for full-function JMS publish/subscribe support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only.

Note: Message-driven beans cannot use the direct listener port for publish/subscribe support. Therefore, any topic connection factory configured with **Port** set to `Direct` cannot be used with message-driven beans.

Data type Enum
Units Not applicable
Default QUEUED
Range
QUEUED
The listener port used for full-function JMS-compliant, publish/subscribe support.
DIRECT
The listener port used for direct TCP/IP connection (non-transactional, non-persistent, and non-durable subscriptions only) for publish/subscribe support.

Component-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for application-managed authentication.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the messaging provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider. For example, the default Windows NT user ID, **Administrator**, is not valid because it contains 13 characters. Therefore, an authentication alias for a WebSphere topic connection factory must specify a user ID no longer than 12 characters.

Container-managed Authentication Alias:

This alias specifies a user ID and password to be used to authenticate connection to the messaging provider for container-managed authentication.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. If the 'DefaultPrincipalMapping' login configuration is used, the associated property is a container-managed authentication alias. This field is used only in the absence of a loginConfiguration on the component resource reference. To define a new alias, see the related item J2EE Connector Architecture (J2C) authentication data entries.

This property provides a list of the J2C authentication data entry aliases that have been defined to WebSphere Application Server. You can select a data entry alias to be used to authenticate the creation of a new connection to the JMS provider.

If you have enabled administrative security for WebSphere Application Server, select the alias that specifies the user ID and password used to authenticate the creation of a new connection to the JMS provider. The use of this alias depends on the resource authentication (res-auth) setting declared in the connection factory resource reference of an application component's deployment descriptors.

Note: User IDs longer than 12 characters cannot be used for authentication with the embedded WebSphere JMS provider. For example, the default Windows NT user ID, **Administrator**, is not valid for use with embedded WebSphere messaging, because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider connection factory must specify a user ID no longer than 12 characters.

Mapping-Configuration Alias:

The module used to map authentication aliases.

The specification of a login configuration and associated properties on the component resource reference determines the container-managed authentication strategy when the res-auth value is Container. This field is used only in the absence of a loginConfiguration on the component resource reference.

This field provides a list of the modules that have been configured on the **Security** → **JAAS Configuration** → **Application Logins Configuration** property. For more information about the mapping configurations, see Java Authentication and Authorization service configuration entry settings.

Data type	Enum
Default	Null

Range

ClientContainer
The client container maps authentication aliases.
WSLogin
The WSLogin module maps authentication aliases.
DefaultPrincipalMapping
The JAAS configuration maps an authentication alias to its userid and password.

Clone Support:

Select this checkbox to enable clone support to allow the same durable subscription across topic clones.

Data type Enum
Default Cleared
Range

Selected
Clone support is enabled.
Cleared
Clone support is disabled.

If you select this property, you must also specify a value for the **Client ID** property.

Client ID:

The JMS client identifier used for connections to the queue manager.

Data type String
Range A valid JMS client ID

XA Enabled:

Specifies whether the connection factory is for XA or non-XA coordination of messages and controls if the application server uses XA QCF/TCF. Enable XA if multiple resources are used in the same transaction.

If you clear this checkbox property (for non-XA coordination), the JMS session is still enlisted in a transaction, but uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, this means that only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

For a WebSphere Topic Connection Factory with the **Port** property set to `DIRECT` this property does not apply, and always adopts non-XA coordination.

Data type Checkbox
Default Selected (enabled for XA coordination)
Range

Selected
The connection factory is enabled for XA-coordination of messages
Cleared
The connection factory is not enabled for XA coordination of messages

Recommended	Do not enable XA coordination when the message queue or topic received is the only resource in the transaction. Enable XA coordination when other resources, including other queues or topics, are involved.

Connection pool:

Specifies an optional set of connection pool settings.

Connection pool properties are common to all J2C connectors.

The application server pools connections and sessions with the messaging provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Change the size of the connection pool if concurrent server-side access to the JMS resource exceeds the default value. The size of the connection pool is set on a per queue or topic basis.

Session pool:

An optional set of session pool settings.

This link provides a panel of optional connection pool properties, common to all J2C connectors.

The application server pools connections and sessions with the JMS provider to improve performance. You need to configure the connection and session pool properties appropriately for your applications, otherwise you may not get the connection and session behavior that you want.

Version 5 WebSphere queue destination settings:

Use this panel to view or change the configuration properties of the selected JMS queue destination for point-to-point messaging by WebSphere Application Server Version 5 applications.

A queue destination is used to configure a JMS queue of the default messaging provider for use by WebSphere Application Server Version 5 applications. Connections to the queue are created by the associated V5 Default Messaging WebSphere queue connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Queues**. This displays a list of any existing JMS queue destinations.
5. Click the name of the JMS queue destination that you want to work with.

A JMS queue for use with the internal WebSphere JMS provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the queue is known for administrative purposes within IBM WebSphere Application Server.

To enable applications to use this queue, you must add the queue name to the Queue Names field on the panel for the JMS server that hosts the queue.

Data type String

JNDI name:

The JNDI name that is used to bind the queue into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the queue, for administrative purposes

Data type String
Default Null

Category:

A category used to classify or group this queue, for your IBM WebSphere Application Server administrative records.

Data type String

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum

Default APPLICATION DEFINED

Range

APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.

NON PERSISTENT
Messages on the destination are not persistent.

PERSISTENT
Messages on the destination are persistent. When a persistent message is put to a queue, all of the message data is written to the messaging log (under the *embedded_messaging_install*\log directory) to make recovery of the message possible.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum

Default APPLICATION DEFINED

Range

APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.

QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.

SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property.*If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer

Units Message priority level

Default 0

Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum

Default APPLICATION DEFINED

Range

APPLICATION DEFINED
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

UNLIMITED
Messages on this queue have no expiry timeout, so those messages never expire.

SPECIFIED
The expiry timeout for messages on this queue is defined by the **Specified expiry** property.*If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Version 5 WebSphere topic destination settings:

Use this panel to browse or change the configuration properties of the selected JMS topic destination for publish/subscribe messaging by WebSphere application server Version 5 applications.

A WebSphere topic destination is used to configure the properties of a JMS topic for the default messaging provider on a Version 5 node. Connections to the topic are created by the associated topic connection factory.

To view this page, use the administrative console to complete the following steps:
1. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
2. **(Optional)** In the content pane, change the **Scope** setting to the level at which JMS resources are visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource. However, the JMS resource has only the subset of properties that apply to WebSphere Application Server Version 5. If you want to define a JMS resource at Cell level for use on non-Version 5 nodes, you should define the JMS resource for the Version 6 default messaging provider.
3. In the content pane, click the name of the V5 default messaging provider that you want to support the JMS destination.
4. Under Additional Resources, click **Topics**. This displays a list of any existing JMS topic destinations.
5. Click the name of the JMS topic destination that you want to work with.

A JMS topic destination for use with the Version 5 default messaging provider has the following properties.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the messaging provider settings panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The JNDI name that is used to bind the topic into the application server's name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the topic, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Category:

A category used to classify or group this topic, for your IBM WebSphere Application Server administrative records.

Data type String

Topic:

The name of the topic as defined to the JMS provider.

Data type String
Default Null
Range The topic value can be dot notation and include wildcard characters.

Persistence:

Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application

Data type Enum
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
Messages on the destination have their persistence defined by the application that put them onto the queue.
NON-PERSISTENT
Messages on the destination are not persistent.
PERSISTENT
Messages on the destination are persistent.

Priority:

Whether the message priority for this destination is defined by the application or the **Specified priority** property

Data type Enum
Units Not applicable
Default APPLICATION DEFINED
Range
APPLICATION DEFINED
The priority of messages on this destination is defined by the application that put them onto the destination.
QUEUE DEFINED
[WebSphere MQ destination only] Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
SPECIFIED
The priority of messages on this destination is defined by the **Specified priority** property. *If you select this option, you must define a priority on the **Specified priority** property.*

Specified priority:

If the **Priority** property is set to `Specified`, type here the message priority for this queue, in the range 0 (lowest) through 9 (highest)

If the **Priority** property is set to `Specified`, messages sent to this queue have the priority value specified by this property.

Data type Integer
Units Message priority level
Default 0
Range 0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this queue is defined by the application or the **Specified expiry** property, or messages on the queue never expire (have an unlimited expiry timeout)

Data type Enum
Units Not applicable
Default APPLICATION DEFINED

Range

APPLICATION DEFINED
The expiry timeout for messages on this queue is defined by the application that put them onto the queue.

UNLIMITED
Messages on this queue have no expiry timeout, so those messages never expire.

SPECIFIED
The expiry timeout for messages on this queue is defined by the **Specified expiry** property. *If you select this option, you must define a timeout on the **Specified expiry** property.*

Specified expiry:

If the **Expiry timeout** property is set to `Specified`, type here the number of milliseconds (greater than 0) after which messages on this queue expire

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Configuring Version 5 default JMS resources

Use the following tasks to configure the JMS connection factories and destinations WebSphere Application Server Version 5 applications.

About this task

You only need to complete these tasks if you have WebSphere application server Version 5 applications that need to use JMS resources provided by the default messaging provider. Such JMS resources are maintained as *V5 Default Messaging* resources.

- Configuring a connection for Version 5 default messaging
- Configuring a Version 5 default JMS queue connection factory
- Configuring a Version 5 default JMS topic connection factory
- Configuring a Version 5 default JMS queue destination
- Configuring a Version 5 default JMS topic destination

Configuring a connection for Version 5 default messaging:

Use this task to configure a connection to Version 5 default messaging. This task is provided to help you manually migrate nodes from v5 to v6. If you use the supplied tools to migrate existing v5 nodes to v6, the jmsserver, appserver and port that you create with this task are defined automatically.

About this task

To configure a connection for Version 5 default messaging, use the administrative console to complete the following steps:

1. Create an application server with the name jmsserver (only one of these can exist on each node). In the navigation pane, expand **Servers** → **Application servers**.

2. On the new server, define a new JMSSERVER_QUEUED_ADDRESS port with the host set to the v6 server host, and the port set to the same as the SIB_MQ_ENDPOINT_ADDRESS of the server which is a member of your bus. The appserver called jmsserver does not actually have to be started; it is only used for looking up the port address.
3. Define a queue connection factory and queue at the Node or Cell scope. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
4. In the content pane, click the name of the V5 default messaging provider.
5. Define a queue destination on your bus with the same name as your queue defined under v5 default messaging.
6. Define an alias destination on your bus with the same name as your queue defined under v5 default messaging but with WQ_ appended to the front the name. For example, if your queue has the name MyV5Queue, your alias should have the name WQ_MyV5Queue.
7. Point the alias at your queue destination with the correct name. The migration process targets the queue with the WQ_ prefix; defining the alias to point to the real queue helps migration.
8. Define the WebSphere MQ Client Link on your messaging engine on the bus. Keep the WebSphere MQ channel name WAS.JMS.SVRCONN and set the queue manager name so that it contains your node name. For example, if your node name is MyNode, you would set the queue manager name to WAS_<MyNode>_jmsserver. Now set the queue manager name to the same; in this example it would be WAS_MyNode_jmsserver. The WebSphere MQ Client Link will show a status of inactive, this is normal.
9. Restart your server so that the new JNDI definitions bind correctly. Your v5 client should now be able to connect to v6 using the host and bootstrap address of the v6 system in the provider url component of the initial context. Your client should now also be able to send messages to the destination on the bus.
10. If you open the Client connections view and click the refresh icon, the hostname of the connecting system is visible whilst the client is connected. In the navigation pane, expand**Buses** → **[bus name]** → **Messaging engines** → **[messaging engine name]** → **WebSphere MQ client links** → **[client link name]** → **Runtime** → **Client connections view**.
11. Click **OK**.
12. Save any changes to the master configuration.
13. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 queue connection factory:

Use this task to browse or change the properties of a JMS queue connection factory for point-to-point messaging with the default messaging provider on a Version 5 node. This task contains an optional step for you to create a new JMS queue connection factory.

About this task

To configure a JMS queue connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a queue connection factory.
3. In the content pane, under Additional Properties, click **Queue connection factories** This displays any existing JMS queue connection factories for the Version 5 messaging provider in the content pane.
4. To browse or change an existing JMS queue connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

Name The name by which this JMS queue connection factory is known for administrative purposes within IBM WebSphere Application Server.

JNDI Name
The JNDI name that is used to bind the JMS queue connection factory into the name space.

c. Click **Apply**. This defines the JMS queue connection factory to WebSphere Application Server, and enables you to browse or change additional properties.

5. Optional: Change properties for the queue connection factory, according to your needs.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 JMS topic connection factory:

Use this task to browse or change a JMS topic connection factory for publish/subscribe messaging by WebSphere Application Server Version 5 applications.

About this task

To configure a JMS topic connection factory for use by WebSphere application server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a topic connection factory.
3. Optional: Change the **Scope** setting to the level at which the JMS topic connection factory is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topic connection factories** This displays any existing JMS topic connection factories for the Version 5 messaging provider in the content pane.
5. To browse or change an existing JMS topic connection factory, click its name in the list. Otherwise, to create a new connection factory, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this JMS topic connection factory is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the JMS topic connection factory into the name space.

   c. Click **Apply**. This defines the JMS topic connection factory to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the topic connection factory, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere queue destination:

Use this task to browse or change the properties of a JMS queue destination for point-to-point messaging by WebSphere Application Server Version 5 applications. This task contains an optional step for you to create a new topic destination.

About this task

To configure a JMS queue destination for use by WebSphere Application Server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Optional: Change the **Scope** setting to the level at which the JMS destination is visible to applications.
3. Select the Version 5 provider for which you want to configure a queue destination.
4. In the content pane, under Additional Properties, click **Queues** This displays any existing queue destinations for the Version 5 default messaging provider in the content pane.
5. To browse or change an existing JMS queue destination, click its name in the list. Otherwise, to create a new queue destination, complete the following steps:
   a. Click **New** in the content pane.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this queue destination is known for administrative purposes within IBM WebSphere Application Server.

      JNDI Name
      The JNDI name that is used to bind the queue destination into the name space.
   c. Click **Apply**. This defines the queue destination to WebSphere Application Server, and enables you to browse or change additional properties.
6. Optional: Change properties for the queue destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To make a queue destination available to applications, host the queue on a JMS server. To add a new queue to a JMS server or to change an existing queue on a JMS server, you define the administrative name of the queue to the JMS server.
10. To have the changed configuration take effect, stop then restart the application server.

Configuring a Version 5 WebSphere topic destination:

Use this task to browse or change the properties of a JMS topic destination for publish/subscribe messaging by WebSphere application server Version 5 applications.. This task contains an optional step for you to create a new topic destination.

About this task

To configure a JMS topic destination for use WebSphere Application Server Version 5 applications, use the administrative console to complete the following steps:

1. Display the Version 5 default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the Version 5 provider for which you want to configure a topic destination.
3. Optional: Change the **Scope** setting to the level at which the JMS destination is visible to applications. If you define a Version 5 JMS resource at the Cell scope level, all users in the cell can look up and use that JMS resource.
4. In the content pane, under Additional Properties, click **Topics** This displays any existing JMS topic destinations for the Version 5 default messaging provider in the content pane.
5. To browse or change an existing JMS topic destination, click its name in the list. Otherwise, to create a new topic destination, complete the following steps:

a. Click **New** in the content pane.
b. Specify the following required properties. You can specify other properties, as described in a later step.

   Name The name by which this topic destination is known for administrative purposes within IBM WebSphere Application Server.

   JNDI Name
   The JNDI name that is used to bind the topic destination into the name space.

   Topic The name of the topic in the default messaging provider, to which messages are sent.
c. Click **Apply**. This defines the topic destination to WebSphere Application Server, and enables you to browse or change additional properties.

6. Optional: Change properties for the topic destination, according to your needs.
7. Click **OK**.
8. Save any changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Configuring authorization security for a Version 5 default messaging provider

Use this task to configure authorization security for the default messaging provider on a WebSphere Application Server Version 5 node.

About this task

To configure authorization security for the Version 5 default messaging provider complete the following steps.

Note: Security for the Version 5 default messaging provider is enabled when you enable WebSphere Application Server security on the Version 5 node. For more information about enabling security, see Enabling security.

1. Configure authorization settings to access JMS resources owned by the embedded messaging subsystem.

   Authorization to access JMS resources owned by the embedded messaging subsystem is controlled by settings in the *profile_root*\config\cells*your_cell_name*\integral-jms-authorizations.xml file. The settings grant or deny authenticated users access to messaging resources (queues or topics). As supplied, the integral-jms-authorisations.xml file grants the following permissions:
   - Read and write permissions to all queues.
   - Pub, sub, and persist to all topics.

   To configure authorization settings, edit the integral-jms-authorisations.xml file according to the information in this topic and in that file. Please note the file is in Unicode, which requires a binary FTP to the host from a workstation.
2. Edit the queue-admin-userids element to create a list of userids with administrative access to all queues. Administrative access is needed to create queues and perform other administrative activities on queues. For example, consider the following queue-admin-userids section:

   ```
   <queue-admin-userids>
       <userid>adminid1</userid>
       <userid>adminid2</userid>
   </queue-admin-userids>
   ```

   In this example the userids adminid1 and adminid2 are defined to have administrative access to all queues.
3. Edit the queue-default-permissions element to define the default queue access permissions. These permissions are used for queues for which you do not define specific permissions (in queue sections). If this section is not specified, then access permissions exist only for those queues for which you have specifically created queue elements.

For example, consider the following queue-default-permissions element:

```
<queue-default-permissions>
  <permission>write</permission>
</queue-default-permissions>
```

In this example the default access permission for all queues is **write**. This can be overridden for a specific queue by creating a queue element that sets its access permission to **read**.

4. If you want to define specific access permissions for a queue, create a queue element, then define the following elements:

   For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

   In this example for the queue q1, the userid useridr has read permission, the userid useridw has write permission, the userid useridrw has both read and write permissions, and all other userids have no access permissions (<public></public>).

5. Edit topic elements to define the access permissions for publish/subscribe topic destinations.

   For topics, you can grant and deny access permissions. Full permission inheritance is supported on topics. If you do not define specific access permissions for a userid on a specific topic then permissions are inherited first from the public permissions on that topic then from the parent topic. The inheritance of access permissions continues until the root topic from which the root permissions are assumed.

   a. If you want to define default access permissions for the root topic, edit a topic element with an empty name element. If you omit such a topic section, topics have no default topic permissions other than those defined by specific topic elements. For example, consider the following topic element for the root topic:

```
<topic>
   <name></name>
   <public>
     <permission>+pub</permission>
   </public>
 </topic>
```

      In this example, the default access permission for all topics is set to publish. This can be overridden by other topic elements for specific topic names.

   b. If you want to define access permissions for a specific topic, create a topic element with the name for the topic then define the access permissions in the public and authorize elements of the topic element. For example, consider the following topic section:

```
<topic>
  <name>a/b/c</name>
  <public>
    <permission>+sub</permission>
  </public>
  <authorize>
```

```
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>
```

In this example, the subscribe permission is granted to anyone accessing any topic whose name starts with `a/b/c`. Also, the userid useridpub is granted publish permission for any topic whose name starts with `a/b/c`.

6. Save the integral-jms-authorizations.xml file.

Results

If the dynamic update setting is selected, changes to the integral-jms-authorizations.xml file become active when the changed file is saved, so there is no need to stop and restarted the JMS server. If the dynamic update setting is not selected, you need to stop and restart the JMS server to make changes active.

Authorization settings for Version 5 default JMS resources:

Use the integral-jms-authorisations.xml file to view or change the authorization settings for Java Message Service (JMS) resources owned by the default messaging provider on WebSphere Application Server Version 5 nodes.

Authorization to access default JMS resources owned by the default messaging provider on WebSphere Application Server nodes is controlled by the following settings in the *was_install*`\config\cells\`*your_cell_name*`\integral-jms-authorisations.xml` file.

This structure of the settings in integral-jms-authorisations.xml is shown in the following example. Descriptions of these settings are provided after the example. To configure authorization settings, follow the instructions provided in Configuring authorization security for the Version 5 JMS providers

```
<integral-jms-authorizations>

  <dynamic-update>true</dynamic-update>

  <queue-admin-userids>
    <userid>adminid1</userid>
    <userid>adminid2</userid>
  </queue-admin-userids>

  <queue-default-permissions>
    <permission>write</permission>
  </queue-default-permissions>

  <queue>
    <name>q1</name>
    <public>
    </public>
    <authorize>
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
    <authorize>
      <userid>useridw</userid>
      <permission>write</permission>
    </authorize>
  </queue>

  <queue>
    <name>q2</name>
    <public>
      <permission>write</permission>
    </public>
    <authorize>
```

```
      <userid>useridr</userid>
      <permission>read</permission>
    </authorize>
  </queue>

  <topic>
    <name></name>
    <public>
      <permission>+pub</permission>
    </public>
  </topic>

  <topic>
    <name>a/b/c</name>
    <public>
      <permission>+sub</permission>
    </public>
    <authorize>
      <userid>useridpub</userid>
      <permission>+pub</permission>
    </authorize>
  </topic>

</integral-jms-authorizations>
```

dynamic-update: Controls whether or not the JMS Server checks dynamically for updates to this file.
true (Default) Enables dynamic update support.
false Disables dynamic update checking and improves authorization performance.

queue-admin-userids: This element lists those userids with administrative access to all Version 5 default queue destinations. Administrative access is needed to create queues and perform other administrative activities on queues. You define each userid within a separate userid sub element:
<userid>*adminid***</userid>**
Where *adminid* is a user ID that can be authenticated by IBM WebSphere Application Server.

queue-default-permissions: This element defines the default queue access permissions that are assumed if no permissions are specified for a specific queue name. These permissions are used for queues for which you do not define specific permissions (in queue elements). If this element is not specified, then no access permissions exist unless explicitly authorized for individual queues.

You define the default permission within a separate permission sub element:
<permission>*read-write***</permission>**
Where *read-write* is one of the following keywords:
read By default, userids have read access to Version 5 default queue destinations.
write By default, userids have write access to Version 5 default queue destinations.

queue: This element contains the following authorization settings for a single queue destination:
name The name of the queue.
public The default public access permissions for the queue. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the queue.

You define each default permission within a separate permission element.
authorize
The access permissions for a specific userid. Within each authorize element, you define the following elements:
userid The userid that you want to assign a specific access permission.
permission
An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain the keyword read or write to define the access permission.

For example, consider the following queue element:

```
<queue>
  <name>q1</name>
  <public>
  </public>
  <authorize>
    <userid>useridr</userid>
    <permission>read</permission>
  </authorize>
  <authorize>
    <userid>useridw</userid>
    <permission>write</permission>
  </authorize>
  <authorize>
    <userid>useridrw</userid>
    <permission>read</permission>
    <permission>write</permission>
  </authorize>
</queue>
```

topic: This element contains the following authorization settings for a single topic destination:

Each topic element has the following sub elements:

name The name of the topic, without wildcards or other substitution characters.

public The default public access permissions for the topic. This is used only for those userids that have no specific authorize element. If you leave this element empty, or do not define it at all, only those userids with authorize elements can access the topic.

You define each default permission within a separate permission element.

authorize

The access permissions for a specific userid. Within each authorize element, you define the following elements:

userid The userid that you want to assign a specific access permission.

permission

An access permission for the associated userid.

You define each permission within a separate permission element. Each permission element can contain one of the following keywords to define the access permission:

+pub Grant publish permission

+sub Grant subscribe permission

+persist

Grant persist permission

-pub Deny publish permission

-sub Deny subscribe permission

-persist

Deny persist permission

Administering listener ports and activation specifications for message-driven beans

Use these tasks to manage the listener ports and activation specifications used by message-driven beans (MDBs). If the JMS provider is implemented as a J2EE Connector Architecture (JCA) resource adapter use an activation specification, otherwise use a listener port. These tasks are supplementary to the tasks of administering resource adapters, and JMS provider resources.

About this task

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

Use the WebSphere Application Server administrative console to configure the following resources for message-driven beans:

- J2C activation specifications for JCA 1.5-compliant message-driven beans. Activation specifications must be provided when the application's resources are configured using the default messaging provider or any generic J2C Resource Adapter that supports inbound messaging.
- The message listener service, listener ports, and listeners for EJB 2.0 message-driven beans deployed against listener ports. Listener ports must be provided when using the following JMS providers: V5 Default Messaging, or Generic.

Listener port or activation specification?

WebSphere Application Server Version 6 or later supports the enhancements made to the EJB 2.1 and Java EE specifications that allow any resource that has a JCA 1.5 adapter to trigger an MDB (formerly only JMS resources could trigger MDBs). WebSphere Application Server Version 5 and EJB 2.0 used the listener port which specified a JMS connection factory and a destination. This allowed a pool of MDB instances to connect to the messaging system and listen to the designated destination. EJB 2.1 specifies an additional requirement of an MDB working with a JCA adapter using an activation specification.

If you are developing WebSphere Application Server Version 6 or later applications, you should consider the following questions:

- Should I use a listener port or an activation specification?
- Should I convert my existing listener ports to activation specifications?
- Will listener ports eventually not be supported anymore?

Here are some guidelines to help you decide:

- If you are using J2EE 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used, so you do not have to make a decision. WebSphere Application Server Version 4 uses message beans, but these are not MDBs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server Version 5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server Version 5 uses listener ports to associate MDB classes with their JMS destinations.
- If you are using Java EE and EJB 2.1 with WebSphere Application Server Version 6 or later and not using JMS, you must use activation specifications. A connector MDB uses JCA to access its resources, so the connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.
- If you are using Java EE and EJB 2.1 with WebSphere Application Server Version 6 or later, the decision depends on whether your JMS provider API is implemented with JCA. In Java EE, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your MDB is a JMS MDB that is implemented as a connector MDB, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

To summarize: in WebSphere Application Server Version 6 or later, the decision on whether or not to use a listener port or an activation specification depends on the following scenarios:

- whether you upgrade your EJB 2.0 MDBs to EJB 2.1
- whether you want your EJB 2.1 MDB to use a JCA adapter

- whether you access your JMS provider via a JCA adapter.

If you have JMS API implementations that do not use JCA, WebSphere Application Server requires listener ports; if all JMS API implementations use JCA, listener ports are no longer required.

You can update the configuration data at any time, but some updates only take effect when the appropriate server is next started.

For additional information about administering support for message-driven beans, see the following topics:
- Configuring a JMS activation specification for MDBs used by the default messaging provider
- "Configuring a J2C activation specification" on page 1728
- "Configuring a J2C administered object" on page 1732
- Configuring message listener resources for EJB 2.0 message-driven beans

Configuring a J2C activation specification

Use this task to configure a J2EE Connector (J2C) activation specification used to deploy message-driven beans with an external resource adapter.

About this task

Use this task if you want to use a message-driven bean as a listener on a Java Connector Architecture (JCA) 1.5 resource adapter other than the default messaging JMS provider.

You can create or modify a J2C activation specification under an installed resource adapter at the cell, node, or server scope. You can select the message listener type from those provided by the given resource adapter.

Configuring a J2C activation specification offers two distinct advantages:
- The activation specification configuration information can be shared among multiple message-driven beans across multiple applications.
- Updates to the configuration properties can be made without the need to redeploy the application.

The following guidelines show which scenarios use activation specifications or listener ports:
- If you are using J2EE 1.2 and Enterprise JavaBeans (EJB) 1.1 with WebSphere Application Server Version 4, message-driven beans are not used so you do not need listener ports or activation specifications. WebSphere Application Server Version 4 uses message beans, but these are not message-driven beans or enterprise beans.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server Version 5, you must use listener ports. The message-driven beans are JMS message-driven beans that implement MessageListener, and there is no JCA support. WebSphere Application Server Version 5 uses listener ports to associate message-driven bean classes with their JMS destinations.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server Version 6, you must use activation specifications. A connector message-driven bean uses JCA to access its resources, so the connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of message-driven beans from EJB 2.0 to EJB 2.1.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server Version 6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your message-driven bean is a JMS message-driven bean that is implemented as a connector message-driven bean, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 message-driven bean in the same way as you would configure an EJB 2.0 message-driven bean, which in WebSphere Application Server is to use a listener port.

To configure a J2C activation specification for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new activation specification.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. Optional: Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Activation specifications heading, click **J2C Activation Specifications**. This lists any existing J2C activation specifications for the external resource adapter in the content pane.
4. Display the properties of the J2C activation specification. If you want to display an existing J2C activation specification, click one of the names listed.

   Alternatively, if you want to create a new J2C activation specification, click **New**, then specify the following required properties:

   Name Type the name by which the activation specification is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

   Message listener type
   Select the message listener type that this activation specification instance should support. This list is based on the deployment descriptor of the external resource adapter.

   Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new activation specification, you may need to click **Apply** before this custom property selection is available.
5. Specify properties for the activation specification, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Activation Specifications collection:

This page contains a list of J2C activation specifications for a resource adapter configuration and is used to create new J2C activation specifications, to select J2C activation specifications for configuration changes, or to delete J2C activation specifications.

Activation specification definitions and classes are provided by a resource adapter when it is installed. Using this information, the administrator can create and configure J2C activation specifications with JNDI names that are then available for applications to use. The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

The following guidelines show which scenarios use activation specifications or listener ports:

- If you are using Java 2 Platform, Enterprise Edition (J2EE) 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used so you do not need listener ports or activation specifications. WebSphere Application Server v4 uses message beans, but these are not MDbs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server v5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server v5 uses listener ;ports to associate MDB classes with their JMS destinations.

- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, you must use activation specifications. A connector MDB uses JCA to access its resources, so the connector must therefore be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can now be implemented with the JCA 1.5 API. If so, your MDB is a JMS MDB that is implemented as a connector MDB, and must therefore be configured with an activation specification. If not, this is the same JMS situation as for J2EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C activation specifications**.
- **Resources** → **Resource Adapters** → **J2C activation specifications**.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Message Listener Type:

The Message Listener Type that is used by this activation specification.

The list of available classes is provided by the resource adapter.

Data type String

J2C Activation Specifications settings:

Use this page to specify the settings for a J2C activation specification.

The resource adapter uses a J2C activation specification to configure a specific endpoint instance. Each application configuring one or more endpoints must specify the resource adapter that sends messages to the endpoint. The application must use the activation specification to provide the configuration properties related to the processing of the inbound messages.

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C activation specifications** → ***activation_specification***.
- **Resources** → **Resource Adapters** → **J2C activation specifications** → ***activation_specification***.

Scope:

Specifies the scope of the resource adapter that supports this activation specification. Only applications that are installed within this scope can use this activation specification.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this activation specification.

Name:

Specifies the display name of the J2C activation specification instance.

A string with no spaces meant to be a meaningful text identifier for the J2C activation specification. Name is required

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the J2C activation specification instance.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Description:

A free-form text string to describe the J2C activation specification instance.

Data type String

Authentication alias:

This optional field is used to bind the J2C activation specification to an authentication alias (configured through the security JAAS screens).

This alias is used to access a user name and password that are set on the configured J2C activation specification. This field is only meaningful if the J2C activation specification you are configuring has a UserName and Password field.

If you have defined security domains in the application server, you can click **Browse...** to select a J2C authentication alias for the resource that you are configuring. Security domains allow you to isolate J2C authentication aliases between servers. The tree view is useful in determining the security domain to which an alias belongs, and the tree view can help you determine the servers that will be able to access each authentication alias. The tree view is tailored for each resource, so domains and aliases are hidden when you cannot use them.

Data type Text

Message Listener Type:

The Message Listener Type used by this activation specification.

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. After you create the activation specification, the field is a read only text field.

Data type Drop-down list or text

Destination JNDIName:

The destination JNDIName field only appears when a message of type javax.jms.Destination with name *Destination* is received.

Configuring a J2C administered object

Use this task to configure a J2C administered object used to configure objects with an external resource adapter.

About this task

To configure a J2C administered object for an external resource adapter, use the administrative console to complete the following steps. This task contains an optional step for you to create a new administered object.

1. Display the external resource adapter. In the navigation pane, click **Resources** → **Resource Adapters** → *adapter_name*. This displays in the content pane a table of properties for the external resource adapter, including links to the types of J2C resources that it provides.
2. Optional: Change the **Scope** setting to the scope level at which the activation specification is to be visible to applications, according to your needs.
3. In the content pane, under the Additional Properties heading, click **J2C Administered Objects**. This lists any existing J2C administered objects for the external resource adapter in the content pane.
4. Display the properties of the J2C administered object. If you want to display an existing J2C administered object, click one of the names listed.

   Alternatively, if you want to create a new J2C administered object, click **New**, then specify the following required properties:

   Name Type the name by which the J2C administered object is known for administrative purposes. The JNDI name is automatically generated based on the value for the Name property.

   Administered object class
   Select the administered object class that this instance should support. This list is based on the deployment descriptor of the external resource adapter.

Depending on the external resource adapter, there can be additional required properties that need to be supplied. To provide values for these properties, click **Custom properties**. When creating a new administered object, you may need to click **Apply** before this custom property selection is available.

5. Specify properties for the administered object, according to your needs .
6. Click **OK**.
7. Save your changes to the master configuration.

J2C Administered Objects collection:

Use this page to specify administered object settings for a Resource Adapter.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C administered objects**
- **Resources** → **Resource Adapters** → **Resource adapters** → **J2C administered objects**

Name:

Specifies display name assigned to this administered object.

Data type String

JNDI Name:

Specifies the JNDI name of the administered object.

Data type String

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Description:

Specifies a description for the administered object.

Data type String

Administered object class:

Specifies the Administered Object class that is associated with this J2C administered object. This class must be one that is provided by the resource adapter.

Data type String

J2C Administered Object settings:

Use this page to specify the settings for an administered object.

Administered object definitions and classes are provided by a resource adapter when you install it. Using this information, the administrator can create and configure J2C administered objects with JNDI names that are then available for applications to use. Some messaging styles may need applications to use special administered objects for sending and synchronously receiving messages (through connection objects using messaging style specific APIs). It is also possible that administered objects may be used to perform transformations on an asynchronously received message in a message provider-specific way. Administered objects can be accessed by a component by using either a resource environment reference or a message destination reference (preferred).

You can access this administrative console page in one of two ways:

- **Resources** → **Resource Adapters** → **Resource adapters** → ***resource_adapter*** → **J2C administered objects** → ***J2C_administered_object***
- **Resources** → **Resource Adapters** → **Resource adapters** → **J2C administered objects** → ***J2C_administered_object***

Scope:

Specifies the scope of the resource adapter that supports this administered object. Only applications that are installed within this scope can use this object.

Data type String

Provider:

Specifies the resource adapter that encapsulates the appropriate classes for this administrative object.

Data type String

Name:

Specifies the name of the J2C administered object instance.

A string with no spaces meant to be a meaningful text identifier for the administered object. This name is required.

Data type String

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name that this administered object is bound under.

The JNDI name is required. If you do not specify one, it is created from the Name field. If not specified, the JNDI name defaults to *eis/[name]*

Data type String

Description:

Specifies a text description of the J2C administered object instance.

Data type String

Administered object class:

For new objects, the list of available classes is provided by the resource adapter in a drop-down list. You can only select classes from this list.

After you create the administered object, you cannot modify the administered object class; it is read only.

Data type Class name

Configuring message listener resources for message-driven beans

Use the following tasks to configure resources needed by the message listener service to support message-driven beans for use with a JMS provider that does not have a J2EE Connector Architecture (JCA) 1.5 resource adapter.

About this task

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, such as the default messaging provider that is part of WebSphere Application Server Version 6. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA resources to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server Version 5).

Here are some guidelines on which scenarios use listener ports or activation specifications:

- If you are using J2EE 1.2 and EJB 1.1 with WebSphere Application Server v4, MDBs are not used, so you do not used listener ports or activation specifications because WebSphere Application Server v4 uses message beans, but these are not MDbs or EJBs.
- If you are using J2EE 1.3 and EJB 2.0 with WebSphere Application Server v5, you must use listener ports. The MDBs are JMS MDBs that implement MessageListener, and there is no JCA support. WebSphere Application Server v5 uses listener ports to associate MDB classes with their JMS destinations.
- If you are using J2EE 1.4 and EJB 2.1 with WebSphere Application Server v6, the decision depends on whether your JMS provider API is implemented with JCA. In J2EE 1.4, the JMS 1.1 API can be implemented with the JCA 1.5 API.
  - If your JMS provider API is implemented with JCA, your MDB is a JMS MDB that is implemented as a connector MDB. A connector MDB uses JCA to access its resources, and so the connector must be configured with an activation specification. This is for new bean development, and does not affect the conversion of MDBs from EJB 2.0 to EJB 2.1.
  - If your JMS provider API is not implemented with JCA, you have the same JMS situation as for Java EE 1.3, and you must configure this EJB 2.1 MDB in the same way as you would configure an EJB 2.0 MDB, which in WebSphere Application Server is to use a listener port.

If you want to deploy an enterprise application to use JMS message-driven beans with a JMS provider that does not have a JCA 1.5 resource adapter, refer to the following subtopics:

Configuring the message listener service:

Use this task to configure the properties of the message listener service for an application server, to support message-driven beans deployed against listener ports.

About this task

If the JMS provider does not have a J2EE Connector Architecture (JCA) 1.5 resource adapter, such as the V5 Default Messaging and WebSphere MQ, you must configure JMS message-driven beans against a listener port (as in WebSphere Application Server Version 5).

If you want to deploy an enterprise application to use message-driven beans with listener ports, you can use this task to browse or change the configuration of the message listener service for an application server.

To configure the message listener service for an application server, use the administrative console to complete the following steps:

1. Display the listener service settings page:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
2. Optional: Browse or change the value of properties for the message-driven bean thread pool.
   a. Click **Thread Pool**
   b. Change the following properties, to suit your needs:

      Minimum size
      The minimum number of threads to allow in the pool.

      Maximum size
      The maximum number of threads to allow in the pool.

      Thread inactivity timeout
      The number of milliseconds of inactivity that should elapse before a thread is reclaimed. A value of 0 indicates not to wait and a negative value (less than 0) means to wait forever.

      Note: The administrative console does not allow you to set the inactivity timeout to a negative number. To do this you must modify the value directly in the config.xml file.

      Allow thread allocation beyond maximum thread size
      Select this check box to enable the number of threads to increase beyond the maximum size configured for the thread pool.

   c. Click **OK**.
3. Optional: Specify any of the following optional properties that you need, as **Custom properties** of the message listener service:

   NON.ASF.RECEIVE.TIMEOUT, MQJMS.POOLING.TIMEOUT, MQJMS.POOLING.THRESHOLD, MAX.RECOVERY.RETRIES, and RECOVERY.RETRY.INTERVAL.

   For more information about these custom properties, see Custom Properties.

   To browse or change the properties, complete the following steps:

a. Click **Custom properties**
b. For each custom property, specify a value to suit your needs.
   If you have not specified a property before:
   1) Click **New**.
   2) Type the name of the property.
   3) Type the value of the property.
   4) Click **OK**.
4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the Application Server.

Message listener service:

The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service**

Custom Properties:

An optional set of name and value pairs for custom properties of the message listener service.

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL
- “DYNAMIC.CONFIGURATION.ENABLED” on page 1743

Message listener port collection:

The message listener ports configured in the administrative domain

This panel displays a list of the message listener ports configured in the administrative domain. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. You can use this panel to add new listener ports or to change the properties of existing listener ports.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

To view this administrative console panel, click **Servers** → **Application Servers** → *application_server* → **[Messaging] Message Listener Service** → **Listener Ports**

To manage a listener port, enable the **Select** check box beside the listener port name in the list and click a button:

Button	Resulting action
Convert to activation specification	Opens a wizard that helps you convert the selected listener port to an activation specification.
New	Accesses the panel to configure a new listener port.
Delete	Deletes the selected listener port or ports.
Start	Starts the selected listener port or ports.
Stop	Stops the selected listener port or ports.

For more information about asynchronous messaging, see “Asynchronous messaging in WebSphere Application Server using JMS” on page 266.

Listener port settings:

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean.

Use this panel to view or change the configuration properties of the selected listener port.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Listener Ports** → *listener_port*

Name:

The name by which the listener port is known for administrative purposes.

Data type String
Default Null

Initial state:

The state that you want the listener port to have when the application server is next restarted

Data type Enum
Units Not applicable
Default Started
Range

Started
When the application server is next started, the listener port is started automatically.

Stopped
When the application server is next started, the listener port is not started automatically. If message-driven beans are to use this listener port on the application server, the system administrator must start the port manually or select the Started value of this property then restart the application server.

Description:

A description of the listener port, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Connection factory JNDI name:

The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`.

Data type	String
Default	Null

Destination JNDI name:

The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.

You cannot use a temporary destination for late responses.

Data type	String
Default	Null

Maximum sessions:

Specifies the maximum number of concurrent sessions that a listener can have with the JMS server to process messages.

Each session corresponds to a separate listener thread and therefore controls the number of concurrently processed messages. Adjust this parameter when the server does not fully use the available capacity of the machine and if you do not need to process messages in a specific message order.

Data type	Integer
Units	Sessions
Default	1
Range	1 through 2147483647
Recommended	• If you want to process messages in a strict message order, set the value to 1, so only one thread is ever processing messages. • If you want to process multiple messages simultaneously (known as “message concurrency”), set this property to a value greater than 1. Keep this value as low as possible to prevent overloading client applications. A good starting point for a 100% JMS workload with short transaction times is 2 to 4 sessions per processor. If longer running transactions exist, you may need more sessions, which should be determined by experimentation.

Maximum retries:

The maximum number of times that the listener tries to deliver a message to a message-driven bean instance before the listener is stopped, in the range 0 through 2147483647.

Note: A WebSphere MQ queue has a similar property called the **BackoutThreshold** property. If your listener port is reading from a WebSphere MQ queue, then the retry limit and the behavior when the limit is reached is determined by whichever of these two properties is set to the lower limit:

- If you exceed the WebSphere MQ queue **BackoutThreshold** limit, the message that cannot be delivered is moved to somewhere else by WebSphere MQ (for example, to the WebSphere MQ backout requeue queue or the WebSphere MQ dead letter queue) and the listener port services

the next message on the queue. In this case, WebSphere Application Server might not know that the message has not been delivered successfully.

- If you exceed the listener port **maximum retries** limit, the listener port stops. You then manually intervene to investigate the problem, possibly to remove the message from the WebSphere MQ queue then restart the listener port.

Data type	Integer
Units	Retry attempts
Default	0 (no retries)
Range	0 (no retries) through 2147483647

Maximum messages:

The maximum number of messages that the listener can process in one transaction.

If the queue is empty, the listener processes each message when it arrives. Each message is processed within a separate transaction.

For the WebSphere V5 default messaging provider or WebSphere MQ as the JMS provider, if messages start accumulating on the queue then the listener can start processing messages in batches. For third-party messaging providers, this property value is passed to the JMS provider but the effect depends on the JMS provider.

Data type	Integer
Units	Number of messages
Default	1
Range	1 through 2147483647
Recommended	For the WebSphere default messaging providers or WebSphere MQ as the JMS provider, if you want to process multiple messages in a single transaction, then set this value to more than 1. If messages start accumulating on the queue, then a value greater than 1 enables multiple messages to be batch-processed into a single transaction, and eliminates much of the overhead of transactions on JMS messages. **Note:** • If one message in the batch fails processing with an exception, the entire batch of messages is put back on the queue for processing. • Any resource lock held by any of the interactions for the individual messages are held for the duration of the entire batch. • Depending on the amount of processing that messages need, and if XA transactions are being used, setting a value greater than 1 can cause the transaction to time out. If an XA transaction does time out routinely because processing multiple messages exceeds the transaction timeout, reduce this property to 1 (to limit processing to one message per transaction) or increase your transaction timeout.

Message listener service custom properties:

Use this panel to view or change an optional set of name and value pairs for custom properties of the message listener service.

To view this administrative console page, click **Servers** → **Application Servers** → *application_server* → **[Communications] Messaging** → **Message Listener Service** → **Custom Properties**

You can use the Custom properties page to define the following properties for use by the message listener service.

- NON.ASF.RECEIVE.TIMEOUT
- MQJMS.POOLING.TIMEOUT
- MQJMS.POOLING.THRESHOLD
- MAX.RECOVERY.RETRIES
- RECOVERY.RETRY.INTERVAL
- DYNAMIC.CONFIGURATION.ENABLED

NON.ASF.RECEIVE.TIMEOUT:

The timeout in milliseconds for synchronous message receives performed by message-driven bean listener sessions in the non-ASF mode of operation.

You should set this property to a non-zero value only if you want to enable the non-ASF mode of operation for all message-driven bean listeners on the application server.

The message listener service has two modes of operation, Application Server Facilities (ASF) and non-Application Server Facilities (non-ASF).

- The ASF mode is meant to provide concurrency and transactional support for applications. For publish/subscribe message-drive beans, the ASF mode provides better throughput and concurrency, because in the non-ASF mode the listener is single-threaded.
- The non-ASF mode is mainly for use with third-party messaging providers that do not support JMS ASF, which is an optional extension to the JMS specification. The non-ASF mode is also transactional but, because the path length is shorter than the ASF mode, usually provides improved performance.

  Use non-ASF if:
  - Your third-party messaging provider does not provide JMS ASF support
  - You are using message-driven beans with WebSphere topic connections with the DIRECT port, because the embedded publish/subscribe broker using that port does not support XA transactions or JMS ASF.
  - Message order is a strict requirement

Data type Integer

Units Milliseconds

Default ASF mode (custom property not created)

Range 0 or greater milliseconds

0 non-ASF mode is disabled

1 or more
The timeout in milliseconds for non-ASF message-driven bean listener synchronous session receives

Recommended If a transaction timeout occurs, the message must recycle causing extra work. If you want to use the non-ASF mode, set this property to lower than the transaction timeout, but leave spare at least the maximum duration of your message-driven bean's onMessage() method. For example, if your message-driven bean's onMessage() method typically takes a maximum of 10 seconds, and the transaction timeout is set to 120 seconds, you might set the NON.ASF.RECEIVE.TIMEOUT property to no more than 110000 (110000 milliseconds, that is 110 seconds).

MQJMS.POOLING.TIMEOUT:

The number of milliseconds after which a connection in the pool is destroyed if it has not been used.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes.

Data type	Integer
Units	Milliseconds
Default	5 minutes
Range	

MQJMS.POOLING.THRESHOLD:

The maximum number of unused connections in the pool.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if there are more than ten unused connections in the pool.

Data type	Integer
Units	Number of connections
Default	10
Range	

MAX.RECOVERY.RETRIES:

The maximum number of times that a listener port managed by this service tries to recover from a failure before giving up and stopping. When stopped the associated listener port is changed to the stop state. The interval between retry attempts is defined by the RECOVERY.RETRY.INTERVAL custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Retry attempts
Default	5
Range	0 (no retries) through 2147483647

RECOVERY.RETRY.INTERVAL:

The time in seconds between retry attempts by a listener port to recover from a failure. The maximum number of retry attempts is defined by the MAX.RECOVERY.RETRIES custom property.

A failure can be one of two things:

- An unexpected error has occurred when a listener port tries to get a message from the JMS provider.
- The connection between the application server and the JMS provider has been lost, usually due to a network error.

Data type	Integer
Units	Seconds
Default	60
Range	1 through 2147483647

DYNAMIC.CONFIGURATION.ENABLED:

This property controls whether the application server on which a listener port is created requires to be restarted. Set this property to true to enable dynamic configuration.

Data type	Boolean
Default	False (not selected)

Creating a new listener port:

Use this task to create a new listener port for the message listener service, so that message-driven beans can be associated with the port to retrieve messages.

About this task

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server Version 5), you are recommended to deploy such beans as J2EE Connector Architecture (JCA) 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you want to deploy an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to create a new listener port for a message-driven bean to retrieve messages from.

To create a new listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Click **New**.
3. Specify the following required properties:

   Name The name by which the listener port is known for administrative purposes.

   Connection factory JNDI name
   The JNDI name for the JMS connection factory to be used by the listener port; for example, `jms/connFactory1`

   Destination JNDI name
   The JNDI name for the destination to be used by the listener port; for example, `jms/destn1`.
4. Optional: Change other properties for the listener port, according to your needs.
5. Click **OK**.
6. Save your changes to the master configuration.
7. To have the changed configuration take effect, stop then restart the application server.

Results

If enabled, the listener port is started automatically when a message-driven bean associated with that port is installed.

Configuring a listener port:

Use this task to browse or change the properties of an existing listener port, used by message-driven beans associated with the port to retrieve messages.

About this task

Although you can continue to deploy an EJB 2.0 message-driven bean against a listener port (as in WebSphere Application Server Version 5), you are recommended to deploy such beans as J2EE Connector Architecture (JCA) 1.5-compliant resources and to upgrade them to be EJB 2.1 message-driven beans.

If you have deployed an enterprise application to use EJB 2.0 message-driven beans with listener ports, use this task to browse or change the configuration of a listener port that a message-driven bean retrieves messages from.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

To configure the properties of a listener port, use the administrative console to complete the following steps:

1. Display the collection list of listener ports:
   a. In the navigation pane, select **Servers** → **Application Servers**.
   b. In the content pane, click the name of the application server.
   c. Under Communications, click **Messaging** → **Message Listener Service**.
   d. Click **Listener Ports**.
2. Click the name of the listener port that you want to work with. This displays the properties of the listener port in the content pane.
3. Optional: Change properties for the listener port, according to your needs.
4. Click **OK**.
5. Save any changes to the master configuration.
6. To have a changed configuration take effect, stop then restart the application server.

Deleting a listener port:

Use this task to delete a listener port from the message listener service, to prevent message-driven beans associated with the port from retrieving messages.

About this task

To delete a listener port, use the administrative console to complete the following steps:

1. In the navigation pane, select **Servers-> Application Servers** This displays a table of the application servers in the administrative domain.
2. In the content pane, click the name of the application server. This displays the properties of the application server in the content pane.
3. Under Communications, click **Messaging** → **Message Listener Service** This displays the Message Listener Service properties in the content pane.

4. In the content pane, click **Listener Ports**. This displays a list of the listener ports.
5. In the content pane, select the check box for the listener port that you want to delete.
6. Click **Delete**. This action stops the port (needed to allow the port to be deleted) then deletes the port.
7. To save your configuration, click **Save** on the task bar of the Administrative console window.
8. To have the changed configuration take effect, stop then restart the application server.

Administering listener ports:

Use the following tasks to administer listener ports, which each define the association between a connection factory, a destination, and a message-driven bean.

About this task

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provider, refer to related tasks.

You can use the WebSphere Application Server administrative console to administer listener ports, as described in the following tasks.

Note: If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. You do not normally need to start or stop a listener port manually.

- Adding a new listener port Use this task to create a new listener port, to specify a new association between a connection factory, a destination, and a message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.
- Configuring a listener port Use this task to browse or change the configuration properties of a listener port.
- Starting a listener port Use this task to start a listener port manually.
- Stopping a listener port Use this task to stop a listener port manually.

Starting a listener port:

Use this task to start a listener port on an application server, to enable the listeners for message-driven beans associated with the port to retrieve messages.

About this task

A listener is active, that is able to receive messages from a destination, if the deployed message-driven bean, listener port, and message listener service are all started. Although you can start these components in any order, they must all be in a started state before the listener can retrieve messages.

If configured as enabled, a listener port is started automatically when a message-driven bean associated with that port is installed. However, you can start a listener port manually, as described in this topic.

When a listener port is started, the listener manager tries to start the listeners for each message-driven bean associated with the port. If a message-driven bean is stopped, the port is started but the listener is not started, and remains stopped. If you start a message-driven bean, the related listener is started.

To start a listener port on an application server, use the administrative console to complete the following steps:

1. If you want the listener for a deployed message-driven bean to be able to receive messages at the port, check that the message-driven bean has been started.
2. Display the collection list of listener ports:

a. In the navigation pane, select **Servers** → **Application Servers**.
b. In the content pane, click the name of the application server.
c. Under Communications, click **Messaging** → **Message Listener Service**.
d. Click **Listener Ports**.
3. Select the check box for the listener port that you want to start.
4. Click **Start**.
5. Save your changes to the master configuration.

Stopping a listener port:

Use this task to stop a listener port on an application server, to prevent the listeners for message-driven beans associated with the port from retrieving messages.

About this task

When you stop a listener port as described in this topic, the listener manager stops the listeners for all message-driven beans associated with the port.

To stop a listener port on an application server, use the administrative console to complete the following steps:
1. Display the collection list of listener ports:
a. In the navigation pane, select **Servers** → **Application Servers**.
b. In the content pane, click the name of the application server.
c. Under Communications, click **Messaging** → **Message Listener Service**.
d. Click **Listener Ports**.
2. Select the check box for the listener port that you want to stop.
3. Click **Stop**.
4. Save your changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the application server.

Message-driven beans - listener port components:

The WebSphere Application Server support for message-driven beans deployed against listener ports is based on JMS message listeners and the message listener service, and builds on the base support for JMS.

Note: From WebSphere Application Server Version 7 listener ports are deprecated. For information about the facilities available to aid migration of configuration information from a listener port to an activation specification for use with the Websphere MQ messaging provided, refer to related tasks.

The main components of WebSphere Application Server support for message-driven beans are shown in the following figure and described after the figure:

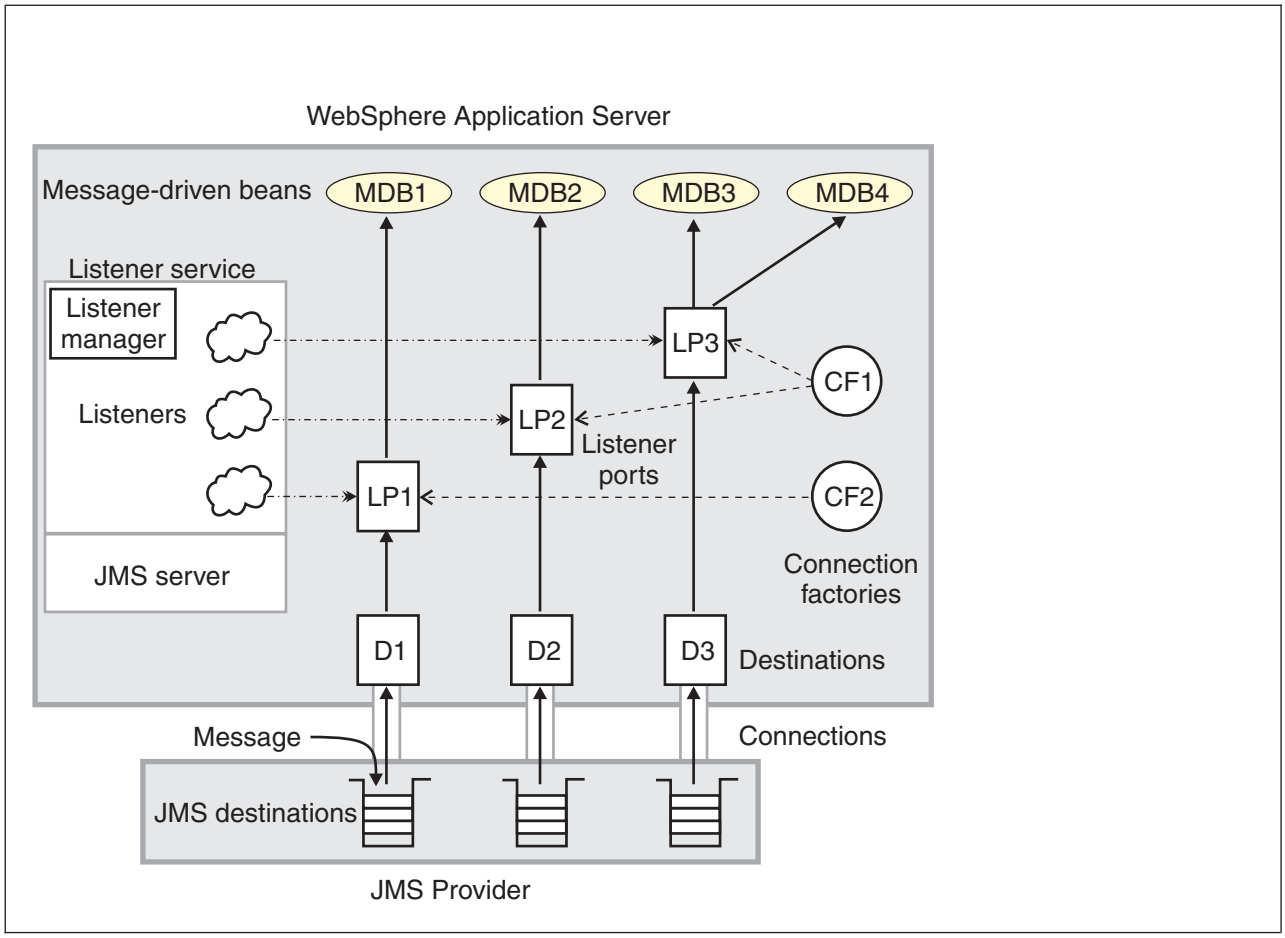


Figure 21. The main components for message-driven beans. This figure shows the main components of WebSphere support for message-driven beans, from JMS provider through a connection to a destination, listener port, then deployed message-driven bean that processes the message retrieved from the destination. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. The other main components are the message listener service, which comprises a listener for each listener port, all controlled by the same listener manager. For more information, see the text that accompanies this figure.

The *message listener service* is an extension to the JMS functions of the JMS provider and provides a *listener manager*, which controls and monitors one or more JMS *listeners*.

Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for publish/subscribe messaging).

A *connection factory* is used to create connections with the JMS provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

A *listener port* defines the association between a connection factory, a destination, and a deployed *message-driven bean*. Listener ports are used to simplify the administration of the associations between these resources.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

When an application server is started, it initializes the listener manager based on the configuration data. The listener manager creates a dynamic session thread pool for use by listeners, creates and starts

listeners, and during server termination controls the cleanup of listener message service resources. Each listener completes several steps for the JMS destination that it is to monitor, including:

- Creating a JMS server session pool, and allocating JMS server sessions and session threads for incoming messages.
- Interfacing with JMS ASF to create JMS connection consumers to listen for incoming messages.
- If specified, starting a transaction and requesting that it is committed (or rolled back) when the EJB method has completed.
- Processing incoming messages by invoking the onMessage() method of the specified enterprise bean.

Important file for message-driven beans

The *server_name*-durableSubscriptions.ser file in the WAS_HOME/temp directory is important for the operation of the WebSphere Application Server messaging service, so should not be deleted.

If you do need to delete the WAS_HOME/temp directory or other files in it, ensure that you preserve the following file:

server_name-**durableSubscriptions.ser**

You should not delete this file, because the messaging service uses it to keep track of durable subscriptions for message-driven beans. If you uninstall an application that contains a message-driven bean, this file is used to unsubscribe the durable subscription.

Learning about messaging with WebSphere Application Server

Use this topic to learn about the use of asynchronous messaging for enterprise applications with WebSphere Application Server.

About this task

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) and J2EE Connector Architecture (JCA) programming interfaces. These interfaces provide a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as messages. For additional information about messaging resources, see: Messaging resources.

- “Types of messaging providers”
- “Styles of messaging in applications” on page 1818
- “JMS interfaces - explicit polling for messages” on page 1819
- “Message-driven beans - automatic message retrieval” on page 1820
- “Configuring JMS resources for the WebSphere MQ messaging provider” on page 1852
- “Message-driven beans - JCA components” on page 1821
- “Asynchronous messaging - security considerations” on page 1825

Types of messaging providers

You can configure any of three main types of Java Message Service (JMS) providers in WebSphere Application Server: The WebSphere Application Server default messaging provider (which uses service integration as the provider), the WebSphere MQ messaging provider (which uses your WebSphere MQ system as the provider) and third-party messaging providers (which use another company’s product as the provider).

Overview

WebSphere Application Server supports JMS messaging through the following providers:

- “Default messaging provider” on page 1815
- “WebSphere MQ messaging provider” on page 1816
- “Third-party messaging provider” on page 1816

Your applications can use messaging resources from any of these JMS providers. The choice of provider is most often dictated by requirements to use or integrate with an existing messaging system. For example, you might already have a messaging infrastructure based on WebSphere MQ. In this case, you can either connect directly using the WebSphere MQ messaging provider, or configure a service integration bus with links to a WebSphere MQ network and then access the bus through the default messaging provider.

Note: For backwards compatibility with earlier releases, WebSphere Application Server Version 7 also includes support for the Version 5 default messaging provider and the Version 6 WebSphere MQ messaging provider. This support enables your applications that still use these resources to communicate with Version 5 and Version 6 nodes in Version 7 mixed cells.

Default messaging provider

If you mainly want to use messaging between applications in WebSphere Application Server, perhaps with some interaction with a WebSphere MQ system, the default messaging provider is the natural choice. This provider is based on service integration technologies and is fully integrated with the WebSphere Application Server runtime environment. To use the default messaging provider, you configure the following resources:

- Configure a connection factory or activation specification to connect your application to a service integration bus.
- Assign a queue or topic to a bus destination on the bus. This topic or queue is then available to any application that can access the bus destination.

A service integration bus comprises messaging engines that run in WebSphere Application Server processes and dynamically connect to one another using dynamic discovery. A messaging application connects to the bus through a messaging engine. Messaging engines use WebSphere Application Server clustering to provide high availability and scalability, and they use the same management framework as the rest of WebSphere Application Server. Bus client applications can run from within WebSphere Application Server (JMS), or run as stand-alone Java clients (using the J2SE Client for JMS) or run as non-Java clients (XMS).

In a pure WebSphere Application Server environment, service integration provides the following benefits:

- An all-Java implementation for JMS within a single server process, yielding good performance for local JMS traffic.
- Direct delivery of messages to a consumer application that is ready to receive them, without first writing the messages to disk.
- Avoidance of copying and serialization of message payloads, if the application is written in accordance with the typical pattern of a messaging application. For more information, see Why and when to pass the JMS message payload by reference.
- Full integration of the administration model with WebSphere Application Server, providing management of single and federated nodes within a cell.
- Full use of the provided WebSphere Application Server infrastructure for tracing and threading, and for a range of communications protocols with a single point of administration.
- Extensive interoperation with WebSphere MQ networks.

There are two ways in which you can connect to a WebSphere MQ system through the default messaging provider:

- Connect a bus to a WebSphere MQ network, using a *WebSphere MQ link*. The WebSphere MQ network appears to the service integration bus as a foreign bus, and the service integration bus appears to WebSphere MQ as another queue manager.

- Connect directly to WebSphere MQ queues located on WebSphere MQ queue managers or (for WebSphere MQ for z/OS) queue-sharing groups, using a *WebSphere MQ server* bus member. Each WebSphere MQ queue is made available at a queue-type destination on the bus.

For more information about these two approaches, see Overview of interoperation with WebSphere MQ.

To configure and manage messaging with the default messaging provider, see Managing messaging with the default messaging provider.

WebSphere MQ messaging provider

If your business also uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominately WebSphere MQ network, choose the WebSphere MQ messaging provider, which allows you to define resources for connecting to any queue manager on the WebSphere MQ network.

WebSphere MQ is characterized as follows:

- Messaging is handled by a network of queue managers, each running in its own set of processes and having its own administration.
- Features such as shared queues (on WebSphere MQ for z/OS) and WebSphere MQ clustering simplify administration and provide dynamic discovery.
- Many IBM and partner products support WebSphere MQ with (for example) monitoring and control, high availability and clustering.
- WebSphere MQ clients can run within WebSphere Application Server (JMS), or almost any other messaging environment using a variety of APIs.

For more information about the WebSphere MQ messaging provider, see "Enhanced features of the WebSphere MQ messaging provider." To configure and manage messaging with this provider, see "Managing messaging with the WebSphere MQ messaging provider" on page 1851. For more information about scenarios and considerations for using WebSphere MQ with WebSphere Application Server, see the white papers and IBM Redbooks publications provided by WebSphere MQ; for example, through the WebSphere MQ library Web page at http://www.ibm.com/software/ts/mqseries/library/.

Third-party messaging provider

You can configure any third-party messaging provider that supports the JMS Version 1.1 unified connection factory. You might want to do this, for example, because of existing investments.

To administer a third-party messaging provider, use the resource adaptor or client supplied by the third party. You can still use the WebSphere Application Server administrative console to administer the JMS connection factories and destinations that are within WebSphere Application Server, but you cannot use the administrative console to administer the JMS provider itself, or any of its resources that are outside of WebSphere Application Server.

To use message-driven beans (MDBs), third-party messaging providers must include Application Server Facility (ASF), an optional feature that is part of the JMS Version 1.1 specification, or use an inbound resource adapter that conforms to the Java EE Connector Architecture (JCA) Version 1.5 specification.

To work with a third-party provider, see "Managing messaging with a third-party messaging provider" on page 1689.

Enhanced features of the WebSphere MQ messaging provider

The WebSphere MQ messaging provider enables WebSphere Application Server applications and clients to connect to and use WebSphere MQ resources in a JMS-compliant manner. For WebSphere Application Server Version 7.0, this provider includes the enhanced features described in this topic.

Overview

For WebSphere Application Server Version 7.0, the WebSphere MQ messaging provider has enhanced administrative options supporting the following functions:

- "WebSphere MQ channel compression"
- "WebSphere MQ client channel definition table"
- "Client channel exits" on page 1818
- "Transport-level encryption using SSL" on page 1818

WebSphere MQ channel compression

Data sent over the network between WebSphere Application Server and WebSphere MQ can be compressed, reducing the amount of data that is transferred. Channel compression can be beneficial in the following situations:

- If a cost is incurred that is proportional to the amount of data transferred over a network. For example, nodes in a network might span a leased line for which a utilization charge is applied.
- If the rate at which messaging data can be transferred across a network is the limiting factor in the performance of an application.
- If compressing the data might reduce the cost of its encryption and decryption.

To use WebSphere MQ channel compression, configure the message compression properties of an existing connection factory or activation specification. For more information, see the appropriate step within "Configuring JMS resources for the WebSphere MQ messaging provider" on page 1852.

For more information, see the WebSphere MQ topic Channel compression.

WebSphere MQ client channel definition table

The client channel definition table reduces the effort required to configure a connection to a queue manager. Your WebSphere MQ administrator can create a single table of all the WebSphere MQ channels supported by queue managers in the enterprise, then in WebSphere Application Server you configure a connection to a queue manager by pointing to a table entry and providing any additional information not already contained within the table.

You can also use the client channel definition table to provide a basic failover capability, by specifying that a connection is attempted against several entries in the table. Each specified entry is tried in turn until a queue manager connection is successfully established.

You can use the client channel definition table, with WebSphere MQ messaging provider activation specifications and connection factories, to select the client channel definition to use when establishing a connection to WebSphere MQ. The table can be configured to select from a number of queue managers, depending on their availability.

When you use a client channel definition table, you must bear in mind the following considerations:

- If your client channel definition table can select from more than one queue manager, you might not be able to recover global transactions. Activation specifications and connection factories that specify a client channel definition table must either do so without ambiguity as to the target queue manager, or must avoid using the resources with applications that enlist in global transactions.
- If your client channel definition table contains entries that reference native WebSphere MQ channel exits, the use of these entries is not supported in the WebSphere Application Server environment.

For more information about client channel definition tables, see the developerWorks article WebSphere MQ V6 Java and JMS clients and the client channel definition table, and the WebSphere MQ topic Client channel definition table.

To use a client channel definition table, point to it when you create a new activation specification or connection factory.

Client channel exits

Client channel exits are pieces of Java code that you develop, and that are executed in the application server at key points during the life cycle of a WebSphere MQ channel. Your code can change the runtime characteristics of the communications link between the WebSphere MQ messaging provider and the WebSphere MQ queue manager.

Note: Only client channel exits written in Java are supported for use within the WebSphere Application Server environment.

For more information about client channel exits, see the WebSphere MQ topic Channel exit programs. For a list of the channel exits that work with the WebSphere MQ messaging provider, see the client connection channel row of the table in the WebSphere MQ topic What are channel exit programs?.

To use client channel exits, configure the client transport properties of an existing connection factory or activation specification.

Transport-level encryption using SSL

Transport-level encryption using SSL is the supported way to configure SSL for JMS resources associated with the WebSphere MQ messaging provider. The SSL configuration is associated with the communication link for the connection factory or activation specification. You either define the SSL information in the connection factory, or your WebSphere MQ administrator defines the SSL information in an associated client channel definition table.

Styles of messaging in applications

This topic describes the ways that applications can use point-to-point and publish/subscribe messaging.

Applications can use the following styles of asynchronous messaging:

Point-to-Point
: Point-to-point applications use *queues* to pass messages between each other. The applications are called point-to-point, because a client sends a message to a specific queue and the message is picked up and processed by a server listening to that queue. It is common for a client to have all its messages delivered to one queue. Like any generic mailbox, a queue can contain a mixture of messages of different types.

Publish/subscribe
: Publish/subscribe systems provide named collection points for messages, called *topics*. To send messages, applications publish messages to topics. To receive messages, applications subscribe to topics; when a message is published to a topic, it is automatically sent to all the applications that are subscribers of that topic. By using a topic as an intermediary, message publishers are kept independent of subscribers.

Both styles of messaging can be used in the same application.

Applications can use asynchronous messaging in the following ways:

One-way
: An application sends a message, and does not want a response. This pattern of use is often referred to as a *datagram*.

Request and response
: An application sends a request to another application and expects to receive a response in return.

One-way and forward
An application sends a request to another application, which sends a message to yet another application.

These messaging techniques can be combined to produce a variety of asynchronous messaging scenarios.

For more information about how such messaging scenarios are used by WebSphere enterprise applications, see the following topics:
- "JMS interfaces - explicit polling for messages"
- "Message-driven beans - automatic message retrieval" on page 1820

For more information about these messaging techniques and the Java Message Service (JMS), see Sun's Java Message Service (JMS) specification documentation (http://developer.java.sun.com/developer/technicalArticles/Networking/messaging/).

For more information about message-driven bean and inbound messaging support, see Sun's Enterprise JavaBeans specification (http://java.sun.com/products/ejb/docs.html).

For information about JCA inbound messaging processing, see Sun's J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).

JMS interfaces - explicit polling for messages

This topic provides an overview of applications that use JMS interfaces to explicitly poll for messages on a destination then retrieve messages for processing by business logic beans (enterprise beans).

WebSphere Application Server supports asynchronous messaging as a method of communication based on the Java Message Service (JMS) programming interfaces. JMS provides a common way for Java programs (clients and Java EE applications) to create, send, receive, and read asynchronous requests, as JMS messages.

The base support for asynchronous messaging using JMS, shown in the following figure, provides the common set of JMS interfaces and associated semantics that define how a JMS client can access the facilities of a JMS provider. This enables WebSphere J2EE applications, as JMS clients, to exchange messages asynchronously with other JMS clients by using JMS destinations (queues or topics).

Applications can use both point-to-point and publish/subscribe messaging (referred to as "messaging domains" in the JMS specification), while supporting the different semantics of each domain.

WebSphere Application Server supports applications that use JMS 1.1 domain-independent interfaces (referred to as the "common interfaces" in the JMS specification). With JMS 1.1, the preferred approach for implementing applications is to use the common interfaces. The JMS 1.1 common interfaces provide a simpler programming model than domain-specific interfaces. Also, applications can create both queues and topics in the same session and coordinate their use in the same transaction.

The common interfaces are also parents of domain-specific interfaces. These domain-specific interfaces (provided for JMS 1.0.2 in WebSphere Application Server Version 5) are supported only to provide inter-operation and backward compatibility with applications that have already been implemented to use those interfaces.

A WebSphere application can use the JMS interfaces to explicitly poll a JMS destination to retrieve an incoming message, then pass the message to a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination.

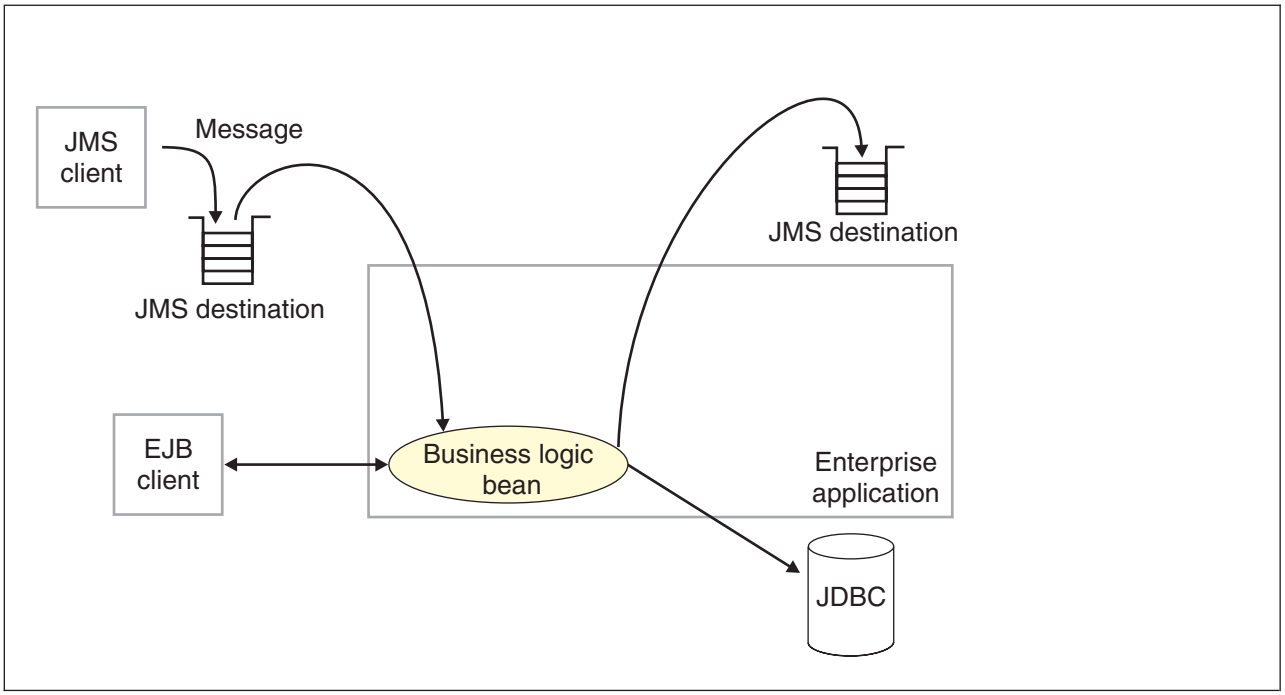


Figure 22. Asynchronous messaging using JMS. This figure shows an enterprise application polling a JMS destination to retrieve an incoming message, which it processes with a business logic bean. The business logic bean uses standard JMS calls to process the message; for example, to extract data or to send the message on to another JMS destination. For more information, see the text that accompanies this figure.

WebSphere applications can use standard JMS calls to process messages, including any responses or outbound messaging. Responses can be handled by an enterprise bean acting as a sender bean, or handled in the enterprise bean that receives the incoming messages. Optionally, this process can use two-phase commit within the scope of a transaction.

WebSphere applications can also use message-driven beans, as described in related topics.

For more details about JMS, see Sun's Java Message Service (JMS) specification documentation.

Message-driven beans - automatic message retrieval

WebSphere Application Server supports the use of message-driven beans as asynchronous message consumers.

Messaging with message-driven beans is shown in the figure Figure 23 on page 1821.

A client sends messages to the destination (or endpoint) for which the message-driven bean is deployed as the message listener. When a message arrives at the destination, the EJB container invokes the message-driven bean automatically without an application having to explicitly poll the destination. The message-driven bean implements some business logic to process incoming messages on the destination.

Message-driven beans can be configured as listeners on a Java EE Connector Architecture (JCA) 1.5 resource adapter or against a listener port (as in WebSphere Application Server Version 5). With a JCA 1.5 resource adapter, message-driven beans can handle generic message types, not just JMS messages. This makes message-driven beans suitable for handling generic requests inbound to WebSphere Application Server from enterprise information systems through the resource adapter. In the JCA 1.5 specification, such message-driven beans are commonly called *message endpoints* or simply *endpoints*.

All message-driven beans must implement the MessageDrivenBean interface. For JMS messaging, a message-driven bean must also implement the message listener interface, javax.jms.MessageListener.

You are recommended to develop a message-driven bean to delegate the business processing of incoming messages to another enterprise bean, to provide clear separation of message handling and business processing. This also enables the business processing to be invoked by either the arrival of incoming messages or, for example, from a WebSphere J2EE client.

Messages arriving at a destination being processed by a message-driven bean have no client credentials associated with them; the messages are anonymous. Security depends on the role specified by the RunAs Identity for the message-driven bean as an EJB component. For more information about EJB security, see Securing enterprise bean applications.

For JMS messaging, message-driven beans can use a JMS provider that has a JCA 1.5 resource adapter, for example the default messaging provider that is part of WebSphere Application Server or the WebSphere MQ messaging provider. With a JCA 1.5 resource adapter, you deploy EJB 2.1 message-driven beans as JCA 1.5-compliant resources, to use a J2C activation specification. If the JMS provider does not have a JCA 1.5 resource adapter, for example the V5 default messaging provider, you must configure JMS message-driven beans against a listener port.

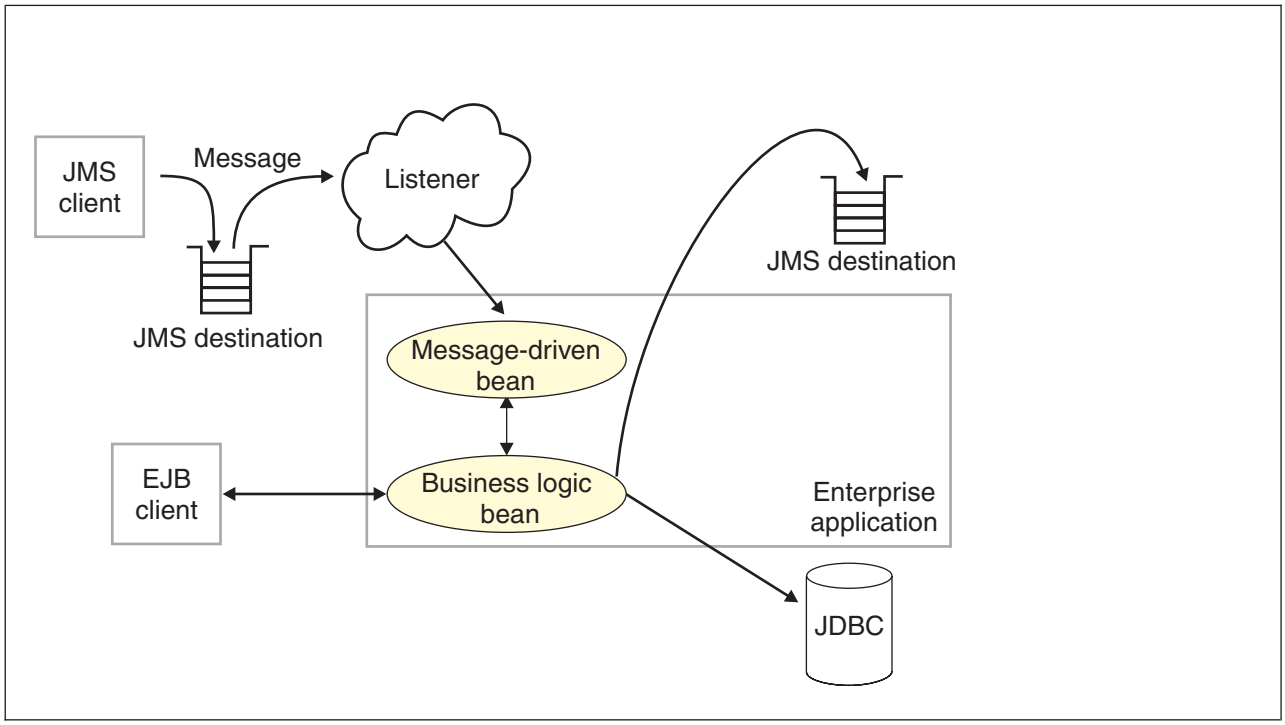


Figure 23. Messaging with message-driven beans. This figure shows an incoming message being passed automatically to the onMessage() method of a message-driven bean that is deployed as a listener for the destination. The message-driven bean processes the message, in this case passing the message on to a business logic bean for business processing. For more information, see the text that accompanies this figure.

Message-driven beans - JCA components

This topic provides an overview of the administrative components that you configure for message-driven beans (MDBs) as listeners on a Java EE Connector Architecture (JCA) 1.5 resource adapter.

Components for a JCA resource adapter

When a resource adapter is installed, it provides definitions and classes for administered objects such as activation specifications. The administrator creates and configures activation specifications with Java™ Naming and Directory Interface (JNDI) names that are then available for applications to use.

The JCA resource adapter uses an activation specification to configure a specific endpoint. Each application that configures one or more endpoints must specify the resource adapter that sends messages to the endpoint

The application uses the activation specification to provide the configuration properties related to the processing of inbound messages.

JMS components used with a JCA messaging provider

MDBs that implement the javax.jms.MessageListener interface can be used for JMS messaging.

An application that uses JMS messaging needs access at runtime to configured objects such as connection factories and destinations.

When the JMS provider is the default JMS provider or the WebSphere MQ messaging provider, the administrator configures these objects for the JMS provider. For example, to configure a JMS activation specification for the WebSphere MQ messaging provider, in the WebSphere Application Server administrative console navigate to **Resources** → **JMS** → **Activation specifications**.

Otherwise the administrator configures these objects for the JMS resource adapter, which connects the application to a JMS provider, by navigating to **Resources** → **Resource Adaptors**.

If the application contains one or more MDBs, the administrator must configure either a JMS activation specification or a message listener port. For JCA-compliant messaging providers, the administrator usually configures an activation specification. But for the WebSphere MQ messaging provider there is a choice; the administrator can configure an activation specification or, for compatibility with previous versions of WebSphere Application Server, the administrator can configure a message listener port.

The JMS activation specification provides the deployer with information about the configuration properties of an MDB related to the processing of the inbound messages. For example, a JMS activation specification specifies the name of the service integration bus to connect to, information about the message acknowledgement modes, message selectors, destination types, and whether or not durable subscriptions are shared across connections with members of a server cluster.

The activation specification identifies a JMS destination by specifying its JNDI name. The MDB acts as a listener on a specific JMS destination.

The JMS destination refers to a service integration bus destination (or WebSphere MQ destination) which the administrator must also configure. For more information about JMS resources and service integration, see Learning about the default messaging provider.

J2C activation specification configuration and use

This topic provides an overview about the configuration and use of J2C activation specifications, used in the deployment of message-driven beans for JCA 1.5 resources.

J2C activation specifications are part of the configuration of inbound messaging support that can be part of a JCA 1.5 resource adapter. Each JCA 1.5 resource adapter that supports inbound messaging defines one or more types of message listener in its deployment descriptor (`messagelistener` in the ra.xml). The message listener is the interface that the resource adapter uses to communicate inbound messages to the message endpoint. A message-driven bean (MDB) is a message endpoint and implements one of the message listener interfaces provided by the resource adapter. By allowing multiple types of message listener, a resource adapter can support a variety of different protocols. For example, the interface javax.jms.MessageListener, is a type of message listener that supports JMS messaging. For each type of message listener that a resource adapter implements, the resource adapter defines an associated activation specification (`activationspec` in the ra.xml). The activation specification is used to set configuration properties for a particular use of the inbound support for the receiving endpoint.

When an application containing a message-driven bean is deployed, the deployer must select a resource adapter that supports the same type of message listener that the message-driven bean implements. As part of the message-driven bean deployment, the deployer needs to specify the properties to set on the J2C activation specification. Later, during application startup, a J2C activation specification instance is created, and these properties are set and used to activate the endpoint (that is, to configure the resource adapter's inbound support for the specific message-driven bean).

Applications with message-driven beans can also specify all, some, or none of the configuration properties needed by the ActivationSpec class, to override those defined by the resource adapter-scoped definition. These properties, specified as activation-config properties in the application's deployment descriptor, are configured when the application is assembled. To change any of these properties requires redeploying the application. These properties are unique to this applications use and are not shared with other message-driven beans. Any properties defined in the application's deployment descriptor take precedence over those defined by the resource adapter-scoped definition. This allows application developers to choose the best defaults for their applications.

WebSphere Application Server activation specification optional binding properties

Binding properties that you can specify for activation specifications to be deployed on WebSphere Application Server.

J2C authentication alias

If you provide values for user name and password as custom properties on an activation specification, you may not want to have those values exposed in clear text for security reasons. WebSphere security allows you to securely define an authentication alias for such cases. Configuration of activation specifications, both as an administrative object and during application deployment, enable you to use the authentication alias instead of providing the user name and password.

If you set the authentication alias field, then you should not set the user name and password custom properties fields. Also, authentication alias properties set as part of application deployment take precedence over properties set on an activation specification administrative object.

Only the authentication alias is ever written to file in an unencrypted form, even for purposes of transaction recovery logging. The security service is used to protect the real user name and password.

During application startup, when the activation specification is being initialized as part of endpoint activation, the server uses the authentication alias to retrieve the real user name and password from security then set it on the activation specification instance.

Destination JNDI name

For resource adapters that support JMS you need to associate javax.jms.Destinations with an activation specification, such that the resource adapter can service messages from the JMS destination. In this case, the administrator configures a J2C Administered Object which implements the javax.jms.Destination interface and binds it into JNDI.

You can configure a J2C Administered Object to use an ActivationSpec class that implements a setDestination(javax.jms.Destination) method. In this case, you can specify the destination JNDI name (that is, the JNDI name for the J2C Administered object that implements the javax.jms.Destination).

A destination JNDI name set as part of application deployment take precedence over properties set on an activation specification administrative object.

During application startup, when the activation specification is being initialized as part of endpoint activation, the server uses the destination JNDI name to look up the destination administered object then set it on the activation specification instance.

Message-driven beans - transaction support

Message-driven beans can handle messages on destinations (or endpoints) within the scope of a transaction.

Transaction handling when using the Message Listener Service with WebSphere MQ JMS

There are three possible cases, based on the MDB deployment descriptor setting you choose: container-managed transaction (required), container-managed transaction (not supported), and bean-managed transaction. For related information see Message-driven bean deployment descriptor properties.

Container-managed transaction (required)

In this situation, the application server starts a global transaction before it reads any incoming message from the destination, and before the onMessage() method of the MDB is invoked by the application server. This means that other EJBs that are invoked in turn by the message, and interactions with resources such as databases can all be scoped inside this single global transaction, within which the incoming message was obtained.

If this application flow completes successfully, the global transaction is committed. If the flow does not complete successfully, (if the transaction is marked for rollback or if a runtime exception occurs), the transaction is rolled back, and the incoming message is rolled back onto the MDB destination.

Container-managed transaction (not supported)

In this situation there is obviously no global transaction, however the JMS provider can still deliver a message from an MDB destination to the application server in a unit of work. You can consider this as a local transaction, because it does not involve other resources in its transactional scope.

The application server acknowledges message delivery on successful completion of the onMessage() dispatch of the MDB (using the acknowledgement mode specified by the assembler of the MDB).

However, the application server does not perform an acknowledge, if an unchecked runtime exception is thrown from the onMessage() method. So, does the message roll back onto the MDB destination (or is it acknowledged and deleted)?

The answer depends on whether or not a syncpoint is used by the WebSphere MQ JMS provider and can vary depending on the operating platform (in particular the z/OS operating platform can impart different behavior here).

If WebSphere MQ establishes a syncpoint around the MDB message consumption in this container-managed transaction (not supported) case, the message is rolled back onto the destination after an unchecked exception.

If a syncpoint is not used, then the message is deleted from the destination after an unchecked exception.

For related information, see the technote 'MDB behavior is different on z/OS than on distributed when getting nonpersistent messages within syncpoint' at http://www.ibm.com/support/docview.wss?uid=swg21231549.

Bean-managed transaction

This situation is similar to the container-managed transaction (not supported) case. Even though there might be a user transaction in this case, any user transaction started within the onMessage dispatch of the MDB does not include consumption of the message from the message-driven bean destination within the transaction scope. To do this, use the container-managed transaction (required) scenario.

Message redelivery

In each of the previous three cases, a message that is rolled back onto the MDB destination is eventually re-dispatched. If the original rollback was due to a temporary system problem, you would expect the re-dispatch of the MDB with this message to succeed on re-dispatch. If, however, the rollback was due to a specific message-related problem, the message would repeatedly be rolled back and re-dispatched. This would be an inefficient use of processing resources.

The application server handles this scenario which is known as a poison message scenario, by tracking the frequency with which a message is dispatched, and by stopping the associated listener port after a specified number of redeliveries has occurred. This is a configurable value on the Maximum Retries property on a listener port. For more information, see “Listener port settings” on page 1738.

Note: A Maximum Retries value of zero stops the listener port after a single failure to successfully complete an onMessage().

Because stopping the listener port stops the processing for all MDBs mapped to that listener port, this solution is rather unspecific. Instead of relying on the WebSphere Application Server message listener service to stop the listener port if a poison message scenario occurs, the other solution is to set up a backout queue (BOQUEUE), and a backout threshold value (BOTHRESH). If you do this, WebSphere MQ handles the poison message. For more information on handling poison messages, see the *WebSphere MQ Using Java* book, which is available from the WebSphere MQ library page at http://www.ibm.com/software/integration/wmq/library/.

Inbound resource adapter transaction handling

An MDB can be configured for bean or container transaction handling. The owner of the resource adapter owner must tell the MDB developer how to set up the MDB for transaction handling.

Asynchronous messaging - security considerations

This topic describes considerations that you should be aware of if you want to use security for asynchronous messaging with WebSphere Application Server.

Security for messaging is enabled only when WebSphere Application Server administrative security is enabled. In this case:

- JMS connections made to the JMS provider are authenticated.
- Access to JMS resources owned by the JMS provider is controlled by access authorizations.
- Requests to create new connections to the JMS provider must provide a user ID and password for authentication.
- The user ID and password do not need to be provided by the application.

If authentication is successful, then the JMS connection is created; if the authentication fails then the connection request is ended.

Standard J2C authentication is used for a request to create a new connection to the JMS provider. If your resource authentication (res-auth) is set to Application, set the alias in the Component-managed Authentication Alias. If the application that tries to create a connection to the JMS provider specifies a user ID and password, those values are used to authenticate the creation request. If the application does not specify a user ID and password, the values defined by the Component-managed Authentication Alias are used. If the connection factory is not configured with a Component-managed Authentication Alias, then you receive a runtime JMS exception when an attempt is made to connect to the JMS provider.

Note:

1. User IDs longer than 12 characters cannot be used for authentication with the Version 5 default messaging provider or WebSphere MQ. For example, the default Windows NT user ID, **Administrator**, is not valid for use because it contains 13 characters. Therefore, an authentication alias for a WebSphere JMS provider or WebSphere MQ connection factory must specify a user ID no longer than 12 characters.
2. If you want to use Bindings transport mode for JMS connections to WebSphere MQ, you set the property **Transport type**=BINDINGS on the WebSphere MQ Queue Connection Factory. You must also choose one of the following options:
   - To use security credentials, ensure that the user specified is the currently logged on user for the WebSphere Application Server process. If the user specified is not the current logged on user for the WebSphere Application Server process, then the WebSphere MQ JMS Bindings authentication throws the error `MQJMS2013 invalid security authentication supplied for MQQueueManager`.
   - Do not specify security credentials. On the WebSphere MQ Connection Factory, ensure that both the **Component-managed Authentication Alias** and the **Container-managed Authentication Alias** properties are not set.

Authorization to access messages stored by the default messaging provider is controlled by authorization to access the service integration bus destinations on which the messages are stored. For information about authorizing permissions for individual bus destinations, see Administering destination roles.

Messaging: Resources for learning

Use the following links to find relevant supplementary information about messaging. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

- Sun's Java Message Service (JMS) specification documentation.

  Provides details about the Java Message Service (JMS).
- Sun's J2EE Connector Architecture specification (http://java.sun.com/j2ee/connector/download.html).

  Provides details about inbound messaging processing using the J2EE Connector architecture.
- J2EE specification

  Provides details about the J2EE specification, including messaging considerations.
- WebSphere MQ Using Java.

  Provides information about using JMS with WebSphere MQ as a messaging provider.
- WebSphere MQ library at http://www.ibm.com/software/integration/wmq/library/

  Provides WebSphere MQ information.
- WebSphere Event Broker library at http://www.ibm.com/software/integration/wbieventbroker/library/

  Provides information about WebSphere MQ Event Broker as a publish/subscribe messaging broker.
- WebSphere Message Broker library at http://www.ibm.com/software/integration/wbimessagebroker/library/

  Provides information about WebSphere Message Broker as a publish/subscribe messaging broker.
- IBM Publications Center

  This Web site provides a wide range of IBM publications, including publications about messaging products.

Configuring messaging with scripting

Use these topics to learn about configuring messaging with the wsadmin tool. You can configure the message listener service, Java Messaging Service settings, queue and connection factories, and WebSphere MQ settings.

About this task

You can use the wsadmin tool to configure various messaging connections and settings.

- Configure the message listener service.

  The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean.
- Configure Java Messaging Service (JMS) providers, destinations, and connections.

  The application server supports asynchronous messaging through the use of a JMS provider and its related messaging system. You can use the wsadmin tool to configure new JMS providers, destinations, and connections.

  A JMS destination is used to configure the properties for the associated messaging provider. Connections to the JMS destination are created by the associated JMS connection factory. A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging.
- Configure queue and topic connection factories for the application server. Use queue and topic connection factories to create connections between providers and destinations. A queue connection factory creats a connection to the associated JMS provider of the JMS queue destinations, for point-to-point messaging. A topic connection factory creates a connection to the associated JMS provider of JMS topic destinations, for publish and subscribe messaging.
- Configure WebSphere MQ settings. A WebSphere MQ server represents either a WebSphere MQ queue manager or a WebSphere MQ queue sharing group. It is used by Service Integration Bus messaging to define properties used for connecting to WebSphere MQ. Setting up a WebSphere MQ server involves using the administrative console to create the server definition, add it to a service integration bus, and create a WebSphere MQ queue type destination.

Configuring the message listener service using scripting

Use scripting to configure the message listener service.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure the message listener service for an application server:

1. Identify the application server and assign it to the server variable:
   - Using Jacl:

     ```
     set server [$AdminConfig getid /Cell:mycell/Node:mynode/Server:server1/]
     ```
   - Using Jython:

     ```
     server = AdminConfig.getid('/Cell:mycell/Node:mynode/Server:server1/')
     print server
     ```

   Example output:

   ```
   server1(cells/mycell/nodes/mynode/servers/server1|server.xml#Server_1)
   ```
2. Identify the message listener service belonging to the server and assign it to the mls variable:
   - Using Jacl:

     ```
     set mls [$AdminConfig list MessageListenerService $server]
     ```
   - Using Jython:

     ```
     mls = AdminConfig.list('MessageListenerService', server)
     print mls
     ```

Example output:

```
(cells/mycell/nodes/mynode/servers/server1|server.xml#MessageListenerService_1)
```

3. Modify various attributes with one of the following examples:
   - This example command changes the thread pool attributes:
     - Using Jacl:

       ```
       $AdminConfig modify $mls {{threadPool {{inactivityTimeout 4000}
       {isGrowable true} {maximumSize 100} {minimumSize 25}}}}
       ```

     - Using Jython:

       ```
       AdminConfig.modify(mls, [['threadPool', [['inactivityTimeout', 4000],
       ['isGrowable',  'true'], ['maximumSize', 100], ['minimumSize', 25]]]])
       ```

   - This example modifies the property of the first listener port:
     - Using Jacl:

       ```
       set lports [$AdminConfig showAttribute $mls listenerPorts]
       set lport [lindex $lports 0]
       $AdminConfig modify $lport {{maxRetries 2}}
       ```

     - Using Jython:

       ```
       lports = AdminConfig.showAttribute(mls, 'listenerPorts')
       cleanLports = lports[1:len(lports)-1]
       lport = cleanLports.split(" ")[0]
       AdminConfig.modify(lport, [['maxRetries', 2]])
       ```

   - This example adds a listener port:
     - Using Jacl:

       ```
       set new [$AdminConfig create ListenerPort $mls {{name my}
       {destinationJNDIName di} {connectionFactoryJNDIName jndi/fs}}]
       $AdminConfig create StateManageable $new {{initialState START}}
       ```

     - Using Jython:

       ```
       new = AdminConfig.create('ListenerPort', mls, [['name', 'my'],
       ['destinationJNDIName', 'di'], ['connectionFactoryJNDIName', 'jndi/fsi']])
       print new
       print AdminConfig.create('StateManageable', new, [['initialState', 'START']])
       ```

     Example output:

     ```
     my(cells/mycell/nodes/mynode/servers/server1:server.xml#ListenerPort_1079471940692)
     (cells/mycell/nodes/mynode/servers/server1:server.xml#StateManageable_107947182623)
     ```

4. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new JMS providers using scripting

You can use the wsadmin tool and scripting to configure a new JMS provider.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new JMS provider:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

```
node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
print node
```

Example output:

```
mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required JMSProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('JMSProvider')
     ```

   Example output:

   ```
   Attribute        Type
   name     String
   externalInitialContextFactory    String
   externalProviderURL               String
   ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name JMSP1]
     set extICF [list externalInitialContextFactory
     "Put the external initial context factory here"]
     set extPURL [list externalProviderURL "Put the external provider URL here"]
     set jmspAttrs [list $name $extICF $extPURL]
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSP1']
     extICF = ['externalInitialContextFactory',
     "Put the external initial context factory here"]
     extPURL = ['externalProviderURL', "Put the external provider URL here"]
     jmspAttrs = [name, extICF, extPURL]
     print jmspAttrs
     ```

   Example output:

   ```
   {name JMSP1} {externalInitialContextFactory {Put the external
   initial context factory here }} {externalProviderURL
   {Put the external provider URL here}}
   ```

4. Create the JMS provider:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig create JMSProvider $node $jmspAttrs]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.create('JMSProvider', node, jmspAttrs)
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new JMS destinations using scripting

You can use scripting and the wsadmin tool to configure a new JMS destination.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new JMS destination:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```
   - Using Jython:
     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:
   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required GenericJMSDestination
     ```
   - Using Jython:
     ```
     print AdminConfig.required('GenericJMSDestination')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   jndiName    String
   externalJNDIName  String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name JMSD1]
     set jndi [list jndiName jms/JMSDestination1]
     set extJndi [list externalJNDIName jms/extJMSD1]
     set jmsdAttrs [list $name $jndi $extJndi]
     ```
   - Using Jython:
     ```
     name = ['name', 'JMSD1']
     jndi = ['jndiName', 'jms/JMSDestination1']
     extJndi = ['externalJNDIName', 'jms/extJMSD1']
     jmsdAttrs = [name, jndi, extJndi]
     print jmsdAttrs
     ```

   Example output:
   ```
   {name JMSD1} {jndiName jms/JMSDestination1} {externalJNDIName jms/extJMSD1}
   ```
4. Create generic JMS destination:
   - Using Jacl:
     ```
     $AdminConfig create GenericJMSDestination $newjmsp  $jmsdAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('GenericJMSDestination', newjmsp,  jmsdAttrs)
     ```

   Example output:
   ```
   JMSD1(cells/mycell/nodes/mynode|resources.xml#GenericJMSDestination_1)
   ```
5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new JMS connections using scripting

Use scripting and the wsadmin tool to configure a new JMS connection.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new JMS connection:

1. Configure a connection pool for your generic connection factories.

   Because Java 2 Connector (J2C) manages the generic connection factories, you must configure a connection pool to indicate the policy for connection management by J2C. The following example commands configure a connection pool in your environment:

   - Using Jacl:

     ```
     set connectionPool [$AdminConfig create ConnectionPool $yourGenericCF {} connectionPool]
     set sessionPool [$AdminConfig create ConnectionPool $yourGenericCF {} sessionPool]
     ```

   - Using Jython:

     ```
     connectionPool = AdminConfig.create('ConnectionPool', '[yourGenericCF {}, connectionPool]')
     sessionPool = AdminConfig.create('ConnectionPool', '[yourGenericCF {}, sessionPool]')
     ```

2. Identify the parent ID:

   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:myNode/JMSProvider:JMSP1]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

3. Get required attributes:

   - Using Jacl:

     ```
     $AdminConfig required GenericJMSConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('GenericJMSConnectionFactory')
     ```

   Example output:

   ```
   Attribute       Type
   name       String
   jndiName    String
   externalJNDIName  String
   ```

4. Set up required attributes:

   - Using Jacl:

     ```
     set name [list name JMSCF1]
     set jndi [list jndiName jms/JMSConnFact1]
     set extJndi [list externalJNDIName jms/extJMSCF1]
     set jmscfAttrs [list $name $jndi $extJndi]
     ```

     Example output:

     ```
     {name JMSCF1} {jndiName jms/JMSConnFact1} {externalJNDIName jms/extJMSCF1}
     ```

   - Using Jython:

     ```
     name = ['name', 'JMSCF1']
     jndi = ['jndiName', 'jms/JMSConnFact1']
     extJndi = ['externalJNDIName', 'jms/extJMSCF1']
     jmscfAttrs = [name, jndi, extJndi]
     print jmscfAttrs
     ```

     Example output:

     ```
     [[name, JMSCF1], [jndiName, jms/JMSConnFact1], [externalJNDIName, jms/extJMSCF1]]
     ```

5. Create generic JMS connection factory:
   - Using Jacl:
     ```
     $AdminConfig create GenericJMSConnectionFactory $newjmsp $jmscfAttrs
     ```
   - Using Jython:
     ```
     print AdminConfig.create('GenericJMSConnectionFactory', newjmsp, jmscfAttrs)
     ```

   Example output:
   ```
   JMSCF1(cells/mycell/nodes/mynode|resources.xml#GenericJMSConnectionFactory_1)
   ```
6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WebSphere queue connection factories using scripting

You can use scripting and the wsadmin tool to configure new queue connection factories in WebSphere Application Server.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WebSphere queue connection factory:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```
   - Using Jython:
     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:
   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required WASQueueConnectionFactory
     ```
   - Using Jython:
     ```
     print AdminConfig.required('WASQueueConnectionFactory')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   jndiName    String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name WASQCF]
     set jndi [list jndiName jms/WASQCF]
     set mqcfAttrs [list $name $jndi]
     ```

     Example output:
     ```
     {name WASQCF} {jndiName jms/WASQCF}
     ```
   - Using Jython:

```
name = ['name', 'WASQCF']
jndi = ['jndiName', 'jms/WASQCF']
mqcfAttrs = [name, jndi]
print mqcfAttrs
```

Example output:

```
[[name, WASQCF], [jndiName, jms/WASQCF]]
```

4. Set up a template:
   - Using Jacl:

     ```
     set template [lindex [$AdminConfig listTemplates WASQueueConnectionFactory] 0]
     ```

   - Using Jython:

     ```
     lineseparator = java.lang.System.getProperty('line.separator')
     template = AdminConfig.listTemplates('WASQueueConnectionFactory').split(lineseparator)[0]
     print template
     ```

5. Create was queue connection factories:
   - Using Jacl:

     ```
     $AdminConfig createUsingTemplate WASQueueConnectionFactory $v5jmsp $mqcfAttrs $template
     ```

   - Using Jython:

     ```
     AdminConfig.createUsingTemplate('WASQueueConnectionFactory', v5jmsp, mqqcfAttrs, template)
     ```

   Example output:

   ```
   WASQCF(cells/mycell/nodes/mynode|resources.xml#WASQueueConnectionFactory_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WebSphere topic connection factories using scripting

Use scripting and the wsadmin tool to configure new WebSphere topic connection factories.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WebSphere topic connection factory:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:

   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASTopicConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('WASTopicConnectionFactory')
     ```

   Example output:

```
Attribute            Type
name                 String
jndiName             String
port                 ENUM(DIRECT, QUEUED)
```

3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name WASTCF]
     set jndi [list jndiName jms/WASTCF]
     set port [list port QUEUED]
     set mtcfAttrs [list $name $jndi $port]
     ```

     Example output:

     ```
     {name WASTCF} {jndiName jms/WASTCF} {port QUEUED}
     ```

   - Using Jython:

     ```
     name = ['name', 'WASTCF']
     jndi = ['jndiName', 'jms/WASTCF']
     port = ['port', 'QUEUED']
     mtcfAttrs = [name, jndi, port]
     print mtcfAttrs
     ```

     Example output:

     ```
     [[name, WASTCF], [jndiName, jms/WASTCF], [port, QUEUED]]
     ```

4. Create was topic connection factories:
   - Using Jacl:

     ```
     $AdminConfig create WASTopicConnectionFactory $v5jmsp $mtcfAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('WASTopicConnectionFactory', v5jmsp, mtcfAttrs)
     ```

   Example output:

   ```
   WASTCF(cells/mycell/nodes/mynode|resources.xml#WASTopicConnectionFactory_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WebSphere queues using scripting

You can use scripting to configure a new WebSphere queue.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WebSphere queue:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
     ```

   - Using Jython:

     ```
     v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
     print v5jmsp
     ```

   Example output:

   ```
   "WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
   ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required WASQueue
     ```

- Using Jython:

```
print AdminConfig.required('WASQueue')
```

Example output:

```
Attribute       Type
name       String
jndiName    String
```

3. Set up required attributes:
   - Using Jacl:

```
set name [list name WASQ1]
set jndi [list jndiName jms/WASQ1]
set wqAttrs [list $name $jndi]
```

Example output:

```
{name WASQ1} {jndiName jms/WASQ1}
```

   - Using Jython:

```
name = ['name', 'WASQ1']
jndi = ['jndiName', 'jms/WASQ1']
wqAttrs = [name, jndi]
print wqAttrs
```

Example output:

```
[[name, WASQ1], [jndiName, jms/WASQ1]]
```

4. Create was queue:
   - Using Jacl:

```
$AdminConfig create WASQueue $v5jmsp $wqAttrs
```

   - Using Jython:

```
print AdminConfig.create('WASQueue', v5jmsp, wqAttrs)
```

Example output:

```
WASQ1(cells/mycell/nodes/mynode|resources.xml#WASQueue_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new WebSphere topics using scripting

You can configure new WebSphere topics using the wsadmin tool and scripting.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new WebSphere topic:

1. Identify the parent ID:
   - Using Jacl:

```
set v5jmsp [$AdminConfig getid "/Cell:mycell/Node:mynode/JMSProvider:WebSphere JMS Provider/"]
```

   - Using Jython:

```
v5jmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:WebSphere JMS Provider/')
print v5jmsp
```

Example output:

```
"WebSphere JMS Provider(cells/mycell/nodes/mynode|resources.xml#builtin_jmsprovider)"
```

2. Get required attributes:
   - Using Jacl:

```
$AdminConfig required WASTopic
```

- Using Jython:

```
print AdminConfig.required('WASTopic')
```

Example output:

```
Attribute       Type
name       String
jndiName    String
topic        String
```

3. Set up required attributes:
   - Using Jacl:

```
set name [list name WAST1]
set jndi [list jndiName jms/WAST1]
set topic [list topic "Put your topic here"]
set wtAttrs [list $name $jndi $topic]
```

Example output:

```
{name WAST1} {jndiName jms/WAST1} {topic {Put your topic here}}
```

   - Using Jython:

```
name = ['name', 'WAST1']
jndi = ['jndiName', 'jms/WAST1']
topic = ['topic', "Put your topic here"]
wtAttrs = [name, jndi, topic]
print wtAttrs
```

Example output:

```
[[name, WAST1], [jndiName, jms/WAST1], [topic, "Put your topic here"]]
```

4. Create was topic:
   - Using Jacl:

```
$AdminConfig create WASTopic $v5jmsp $wtAttrs
```

   - Using Jython:

```
print AdminConfig.create('WASTopic', v5jmsp, wtAttrs)
```

Example output:

```
WAST1(cells/mycell/nodes/mynode|resources.xml#WASTopic_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring a new connection factory for the WebSphere MQ messaging provider using scripting

You can use scripting to configure a new connection factory for the WebSphere MQ messaging provider.

Before you begin

You can also use the “createWMQConnectionFactory command” on page 1966 to create a connection factory for the WebSphere MQ messaging provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a connection factory for the WebSphere MQ messaging provider:

1. Identify the parent ID:
   - Using Jacl:

```
set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/
JMSProvider:WebSphere MQ JMS Provider/]
```

- Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/
JMSProvider:WebSphere MQ JMS Provider')
print newjmsp
```

Example output:

```
WebSphere MQ JMS Provider(cells/mycell/nodes/mynode|
resources.xml#builtin_mqprovider)
```

2. Get the required attributes:
   - Using Jacl:

```
$AdminConfig required MQConnectionFactory
```

   - Using Jython:

```
print AdminConfig.required('MQConnectionFactory')
```

   Example output:

```
attribute    Type
name         String
jndiName     String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name MQCF]
set jndi [list jndiName jms/MQCF]
set mqcfAttrs [list $name $jndi]
```

     Example output:

```
{name MQCF} {jndiName jms/MQCF}
```

   - Using Jython:

```
name = ['name', 'MQCF']
jndi = ['jndiName', 'jms/MQCF']
mqcfAttrs = [name, jndi]
print mqcfAttrs
```

     Example output:

```
[[name, MQCF], [jndiName, jms/MQCF]]
```

4. Set up a template:
   - Using Jacl:

```
set template [lindex [$AdminConfig listTemplates MQConnectionFactory] 0]
```

   - Using Jython:

```
import java
lineseparator = java.lang.System.getProperty('line.separator')
template =
AdminConfig.listTemplates('MQConnectionFactory').split(lineseparator)[0]
print template
```

   Example output:

```
Example non-XA WMQ ConnectionFactory(templates/
system:JMS-resource-provider-templates.xml
#MQConnectionFactory_3)
```

5. Create a connection factory for the WebSphere MQ messaging provider:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQConnectionFactory
$newjmsp $mqcfAttrs $template
```

   - Using Jython:

```
print AdminConfig.createUsingTemplate('MQConnectionFactory',
newjmsp, mqcfAttrs, template)
```

Example output:

```
MQCF(cells/mycell/nodes/mynode:resources.xml#MQConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Defining a connection factory for the WebSphere MQ messaging provider by supplying custom connection data

The following example creates a connection factory, specifying custom connection data. Due to the default values assumed for the unspecified parameters, applications using this connection factory expect to be co-located with a queue manager installed on the same node.

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
wsadmin>$AdminTask createWMQConnectionFactory
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name cf1 -jndiName "jms/cf/cf1" -type CF}
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1205322636000)
```

The following example creates an connection factory for which the user must specify and maintain all the parameters used for establishing a connection to WebSphere MQ.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQConnectionFactory("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)", ["-name cf2
 -jndiName 'jms/cf/cf2' -type CF -description 'Must remember to keep each
of these connection factories in sync with the WebSphere MQ queue manager
to which they refer' -qmgrName QM1 -qmgrHostname 192.168.0.22 -qmgrPort 1415
 -qmgrSvrconnChannel QM1.SVRCONN"])
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263601)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQConnectionFactory
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name cf2 -jndiName "jms/cf/cf2" -type CF -description "Must remember to
keep each of these connection factories in sync with the WebSphere MQ queue
manager to which they refer" -qmgrName QM1 -qmgrHostname 192.168.0.22
-qmgrPort 1415 -qmgrSvrconnChannel QM1.SVRCONN}
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263601)
```

Configuring a new queue connection factory for the WebSphere MQ messaging provider using scripting

You can use scripting to configure a new queue connection factory for the WebSphere MQ messaging provider.

Before you begin

You can also use the “createWMQConnectionFactory command” on page 1966 to create a queue connection factory for the WebSphere MQ messaging provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new queue connection factory for the WebSphere MQ messaging provider:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
   ```

2. Get the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MQQueueConnectionFactory
     ```

   - Using Jython:

     ```
     print AdminConfig.required('MQQueueConnectionFactory')
     ```

   Example output:

   ```
   attribute  Type
   name       String
   jndiName   String
   ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name MQQCF]
     set jndi [list jndiName jms/MQQCF]
     set mqqcfAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name MQQCF} {jndiName jms/MQQCF}
     ```

   - Using Jython:

     ```
     name = ['name', 'MQQCF']
     jndi = ['jndiName', 'jms/MQQCF']
     mqqcfAttrs = [name, jndi]
     print mqqcfAttrs
     ```

     Example output:

     ```
     [[name, MQQCF], [jndiName, jms/MQQCF]]
     ```

4. Set up a template:
   - Using Jacl:

     ```
     set template [lindex [$AdminConfig listTemplates MQQueueConnectionFactory] 0]
     ```

   - Using Jython:

     ```
     import java
     lineseparator = java.lang.System.getProperty('line.separator')
     template = AdminConfig.listTemplates('MQQueueConnectionFactory').
     split(lineseparator)[0]
     print template
     ```

   Example output:

   ```
   Example non-XA WMQ QueueConnectionFactory(templates/
   system:JMS-resource-provider-templates.xml
   #MQQueueConnectionFactory_3)
   ```

5. Create a queue connection factory for the WebSphere MQ messaging provider:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQQueueConnectionFactory
$newjmsp $mqqcfAttrs $template
```

- Using Jython:

```
print AdminConfig.createUsingTemplate('MQQueueConnectionFactory',
newjmsp, mqqcfAttrs, template)
```

Example output:

```
MQQCF(cells/mycell/nodes/mynode:resources.xml#MQQueueConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Defining a queue connection factory for the WebSphere MQ messaging provider by supplying custom connection data

The following example creates a queue connection factory, specifying custom connection data.

```
wsadmin>AdminTask.createWMQConnectionFactory('"WebSphere MQ JMS Provider
(cells/EXAMPLECell01|resources.xml#builtin_mqprovider)"', '
[-type QCF -name MQQueueConnectionFactory1 -jndiName
jms/MQQueueConnectionFactory1 -description -qmgrName
QueueManagerName -wmqTransportType BINDINGS_THEN_CLIENT
-qmgrHostname HostName -qmgrSvrconnChannel ServerConnectionChannel ]')
```

The following example creates a queue connection factory, specifying CCDT connection data.

```
wsadmin>AdminTask.createWMQConnectionFactory('"WebSphere MQ JMS Provider
(cells/EXAMPLECell01|resources.xml#builtin_mqprovider)"', '
[-type QCF -name MQQueueConnectionFactory2 -jndiName
jms/MQQueueConnectionFactory2 -description -ccdtUrl
http://ClientChannelDefinitionTableURL -ccdtQmgrName QueueManager ]')
```

Configuring a new topic connection factor for the WebSphere MQ messaging provider using scripting

You can use scripting to configure a new topic connection factory for the WebSphere MQ messaging provider.

Before you begin

You can also use the “createWMQConnectionFactory command” on page 1966 to create a topic connection factory for the WebSphere MQ messaging provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a topic connection factory for the WebSphere MQ messaging provider:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/
     JMSProvider:JMSP1/]
     ```

   - Using Jython:

     ```
     newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/
     JMSProvider:JMSP1')
     print newjmsp
     ```

   Example output:

   ```
   JMSP1(cells/mycell/nodes/mynode:resources.xml#JMSProvider_1)
   ```

2. Get the required attributes:

- Using Jacl:

```
$AdminConfig required MQTopicConnectionFactory
```

- Using Jython:

```
print AdminConfig.required('MQTopicConnectionFactory')
```

Example output:

```
attribute    Type
name      String
jndiName   String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name MQTCF]
set jndi [list jndiName jms/MQTCF]
set mqtcfAttrs [list $name  $jndi]
```

     Example output:

```
{name MQTCF} {jndiName jms/MQTCF}
```

   - Using Jython:

```
name = ['name', 'MQTCF']
jndi = ['jndiName', 'jms/MQTCF']
mqtcfAttrs = [name, jndi]
print  mqtcfAttrs
```

     Example output:

```
[[name, MQTCF], [jndiName, jms/MQTCF]]
```

4. Set up a template:
   - Using Jacl:

```
set template [lindex [$AdminConfig listTemplates MQTopicConnectionFactory] 0]
```

   - Using Jython:

```
import java
lineseparator = java.lang.System.getProperty('line.separator')
template = AdminConfig.listTemplates('MQTopicConnectionFactory').
split(lineseparator)[0]
print template
```

     Example output:

```
Example non-XA WMQ TopicConnectionFactory(templates/
system: JMS-resource-provider-templates.xml
#MQTopicConnectionFactory_5)
```

5. Create a topic connection factory for the WebSphere MQ messaging provider:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQTopicConnectionFactory
$mqjmsp $mqtcfAttrs $template
```

   - Using Jython:

```
print  AdminConfig.createUsingTemplate('MQTopicConnectionFactory',
mqjmsp, mqtcfAttrs, template)
```

     Example output:

```
MQTCF(cells/mycell/nodes/mynode:resources.xml#MQTopicConnectionFactory_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Defining a topic connection factory for the WebSphere MQ messaging provider by supplying custom connection data

The following example creates a topic connection factory, specifying custom connection data.

Using Jython:

```
wsadmin>AdminTask.createWMQConnectionFactory('"WebSphere MQ JMS Provider
(cells/EXAMPLECell01|resources.xml#builtin_mqprovider)"', '
[-type TCF -name MQTopicConnectionFactory1 -jndiName
 jms/MQTopicConnectionFactory1 -description -qmgrName
QueueManagerName -wmqTransportType BINDINGS_THEN_CLIENT
-qmgrHostname HostName -qmgrSvrconnChannel ServerConnectionChannel ]')
```

The following example creates a topic connection factory, specifying CCDT connection data.

Using Jython:

```
wsadmin>AdminTask.createWMQConnectionFactory('"WebSphere MQ JMS Provider
(cells/EXAMPLECell01|resources.xml#builtin_mqprovider)"', '
[-type TCF -name MQTopicConnectionFactory2 -jndiName
jms/MQTopicConnectionFactory2 -description -ccdtUrl
http://ClientChannelDefinitionTableURL -ccdtQmgrName QueueManager ]')
```

Configuring a new queue for the WebSphere MQ messaging provider using scripting

You can use scripting to configure a new queue for the WebSphere MQ messaging provider.

Before you begin

You can also use the “createWMQQueue command” on page 1992 to create a queue for the WebSphere MQ messaging provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new queue for the WebSphere MQ messaging provider:

1. Identify the parent ID:
   - Using Jacl:

```
set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
```

   - Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1') print newjmsp
```

   Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

2. Get the required attributes:
   - Using Jacl:

```
$AdminConfig required MQQueue
```

   - Using Jython:

```
print AdminConfig.required('MQQueue')
```

   Example output:

```
Attribute       Type name       String jndiName    String baseQueueName  String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name MQQ] set jndi [list jndiName jms/MQQ] set baseQN [list baseQueueName "Put the
 base queue name here"] set mqqAttrs [list $name $jndi $baseQN]
```

Example output:

```
{name MQQ} {jndiName jms/MQQ} {baseQueueName {Put the base queue name here}}
```

- Using Jython:

```
name = ['name', 'MQQ'] jndi = ['jndiName', 'jms/MQQ'] baseQN = ['baseQueueName', "Put the base
 queue name here"] mqqAttrs = [name, jndi, baseQN] print mqqAttrs
```

Example output:

```
[[name, MQQ], [jndiName, jms/MQQ], [baseQueueName, "Put the base queue name here"]]
```

4. Set up a template:
   - Using Jacl:

```
set template [lindex [$AdminConfig listTemplates MQQueue] 0]
```

   - Using Jython:

```
import java lineseparator = java.lang.System.getProperty('line.separator') template =
 AdminConfig.listTemplates('MQQueue').split(lineseparator)[0] print template
```

   Example output:

```
Example.JMS.WMQ.Q1(templates/system:JMS-resource-provider- templates.xml#MQQueue_1)
```

5. Create a queue for the WebSphere MQ messaging provider:
   - Using Jacl:

```
$AdminConfig createUsingTemplate MQQueue $newjmsp $mqqAttrs $template
```

   - Using Jython:

```
print AdminConfig.createUsingTemplate('MQQueue', newjmsp, mqqAttrs, template)
```

   Example output:

```
MQQ(cells/mycell/nodes/mynode|resources.xml#MQQueue_1)
```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Defining a queue for the WebSphere MQ messaging provider

```
AdminTask.createWMQQueue('EXAMPLECell01(cells/EXAMPLECell01|cell.xml)', '[-name MQQueue
 -jndiName jms/MQQueue -queueName QueueName -qmgr QueueManager -description ]')
```

Configuring a new topic for the WebSphere MQ messaging provider using scripting

You can use scripting to configure a new topic for the WebSphere MQ messaging provider.

Before you begin

You can also use the “createWMQTopic command” on page 1981 to create a queue for the WebSphere MQ messaging provider.

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new topic for the WebSphere MQ messaging provider:

1. Identify the parent ID:
   - Using Jacl:

```
set newjmsp [$AdminConfig getid /Cell:mycell/Node:mynode/JMSProvider:JMSP1/]
```

   - Using Jython:

```
newjmsp = AdminConfig.getid('/Cell:mycell/Node:myNode/JMSProvider:JMSP1') print newjmsp
```

   Example output:

```
JMSP1(cells/mycell/nodes/mynode|resources.xml#JMSProvider_1)
```

2. Get the required attributes:
   - Using Jacl:

```
$AdminConfig required MQTopic
```

   - Using Jython:

```
print AdminConfig.required('MQTopic')
```

   Example output:

```
Attribute           Type name                String jndiName             String baseTopicName
     String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name MQT] set jndi [list jndiName jms/MQT] set baseTN [list baseTopicName "Put the
 base topic name here"] set mqtAttrs [list $name $jndi $baseTN]
```

     Example output:

```
{name MQT} {jndiName jms/MQT} {baseTopicName {Put the base topic name here}}
```

   - Using Jython:

```
name = ['name', 'MQT'] jndi = ['jndiName', 'jms/MQT'] baseTN = ['baseTopicName', "Put the base
 topic name here"] mqtAttrs = [name, jndi, baseTN] print mqtAttrs
```

     Example output:

```
[[name, MQT], [jndiName, jms/MQT], [baseTopicName, "Put the base topic name here"]]
```

4. Create a topic for the WebSphere MQ messaging provider:
   - Using Jacl:

```
$AdminConfig create MQTopic $newjmsp $mqtAttrs
```

   - Using Jython:

```
print AdminConfig.create('MQTopic', newjmsp, mqtAttrs)
```

   Example output:

```
MQT(cells/mycell/nodes/mynode|resources.xml#MQTopic_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Defining a topic for the WebSphere MQ messaging provider

```
AdminTask.createWMQTopic('DRAPERDCell01(cells/DRAPERDCell01|cell.xml)', '[-name MQQueue
 -jndiName jms/MQQueue -topicName TopicName -qmgr QueueManager -description ]')
```

JCAManagement command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure messaging with scripting. The commands and parameters in the JCA management group can be used to create and manage resource adapters, Java EE Connector Architecture (J2C) activation specifications, administrative objects, connection factories, administrative object interfaces, and message listener types.

The JCAManagement command group for the AdminTask object includes the following commands:

copyResourceAdapter

Use the copyResourceAdapter command to create a Java 2 Connector (J2C) resource adapter under the scope that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-name
Indicates the name of the new J2C resource adapter. This parameter is required.

-scope
Indicates the scope object ID. This parameter is required.

-useDeepCopy
If you set this parameter to `true`, all of the J2C connection factory, J2C activation specification, and J2C administrative objects will be copied to the new J2C resource adapter (deep copy). If you set this parameter to `false`, the objects are not created (shallow copy). The default is `false`.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask copyResourceAdapter $ra [subst {-name newRA -scope $scope}]
```

- Using Jython string:

```
AdminTask.copyResourceAdapter(ra, '[-name newRA -scope scope]')
```

- Using Jython list:

```
AdminTask.copyResourceAdapter(ra, ['-name', 'newRA', '-scope', 'scope'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask copyResourceAdapter {-interactive}
```

- Using Jython:

```
AdminTask.copyResourceAdapter('-interactive')
```

createJ2CActivationSpec

Use the createJ2CActivationSpec command to create a Java 2 Connector (J2C) activation specification under a J2C resource adapter and the attributes that you specify. Use the messageListenerType parameter to indicate the activation specification that is defined for the J2C resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

- messageListenerType
Identifies the activation specification for the J2C activation specification to be created. Use this parameter to identify the activation specification template for the J2C resource adapter that you specify.

-name
Indicates the name of the J2C activation specification that you are creating.

-jndiName
Indicates the name of the Java Naming and Directory Interface (JNDI).

-destinationJndiName
Indicates the name of the Java Naming and Directory Interface (JNDI) of corresponding destination.

-authenticationAlias
Indicates the authentication alias of the J2C activation specification that you are creating.

-description
Description of the created J2C activation specification.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CActivationSpec $ra {-name J2CAct Spec -jndiName eis/ActSpec1
 -messageListenerType javax.jms.MessageListener }
```

- Using Jython string:

```
AdminTask.createJ2CActivationSpec(ra, '[-name J2CActSpec -jndiName eis/ActSpec1
 -messageListenerType javax.jms.MessageListener]')
```

- Using Jython list:

```
AdminTask.createJ2CActivationSpec(ra, ['-name', 'J2CActSpec', '-jndiName', 'eis/ActSpec1',
 '-messageListenerType', 'javax.jms.MessageListener'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CActivationSpec {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CActivationSpec('-interactive')
```

createJ2CAdminObject

Use the createJ2CAdminObject command to create an administrative object under a resource adapter with attributes that you specify. Use the administrative object interface to indicate the administrative object that is defined in the resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

-adminObjectInterface
Specifies the administrative object interface to identify the administrative object for the resource adapter that you specify. This parameter is required.

-name
Indicates the name of the administrative object.

-jndiName
Specifies the name of the Java Naming and Directory Interface (JNDI).

-description
Description of the created J2C admin object.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CAdminObject $ra {-adminObjectInterface fvt.adapter.message.FVTMessageProvider
 -name J2CA01 -jndiName eis/J2CA01}
```

- Using Jython string:

```
AdminTask.createJ2CAdminObject(ra, '[-adminObjectInterface fvt.adapter.message.FVTMessageProvider
 -name J2CA01 -jndiName eis/J2CA01]')
```

- Using Jython list:

```
AdminTask.createJ2CAdminObject(ra, ['-adminObjectInterface',
 'fvt.adapter.message.FVTMessageProvider', '-name', 'J2CA01', '-jndiName', 'eis/J2CA01'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CAdminObject {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CAdminObject('-interactive')
```

createJ2CConnectionFactory

Use the createJ2CConnectionFactory command to create a Java 2 connection factory under a Java 2 resource adapter and the attributes that you specify. Use the connection factory interfaces to indicate the connection definitions that are defined for the Java 2 resource adapter.

Target object

J2C Resource Adapter object ID

Parameters and return values

-connectionFactoryInterface
Identifies the connection definition for the Java 2 resource adapter that you specify. This parameter is required.

-name
Indicates the name of the connection factory.

-jndiName
Indicates the name of the Java Naming and Directory Interface (JNDI).

-description
Description of the created J2C connection factory.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask createJ2CConnectionFactory $ra {-connectionFactoryInterfaces javax.sql.DataSource
-name J2CCF1 -jndiName eis/J2CCF1}
```

- Using Jython string:

```
AdminTask.createJ2CConnectionFactory(ra, '[-connectionFactoryInterfaces javax.sql.DataSource
-name J2CCF1 -jndi Name eis/J2CCF1]')
```

- Using Jython list:

```
AdminTask.createJ2CConnectionFactory(ra, ['-connectionFactoryInterfaces',
 'javax.sql.DataSource', '-name', 'J2CCF1', '-jndiName', 'eis/J2CCF1'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask createJ2CConnectionFactory {-interactive}
```

- Using Jython:

```
AdminTask.createJ2CConnectionFactory('-interactive')
```

listAdminObjectInterfaces

Use the listAdminObjectInterfaces command to list the administrative object interfaces that are defined under the resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None
- Returns: A list of administrative object interfaces.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listAdminObjectInterfaces $ra
```

- Using Jython string:

```
AdminTask.listAdminObjectInterfaces(ra)
```

- Using Jython list:

```
AdminTask.listAdminObjectInterfaces(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listAdminObjectInterfaces {-interactive}
```

- Using Jython:

```
AdminTask.listAdminObjectInterfaces('-interactive')
```

listConnectionFactoryInterfaces

Use the listConnectionFactoryInterfaces command to list all of the connection factory interfaces that are defined under the Java 2 connector resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None
- Returns: A list of connection factory interfaces.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listConnectionFactoryInterfaces $ra
```

- Using Jython string:

```
AdminTask.listConnectionFactoryInterfaces(ra)
```

- Using Jython list:

```
AdminTask.listConnectionFactoryInterfaces(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listConnectionFactoryInterfaces {-interactive}
```

- Using Jython:

```
AdminTask.listConnectionFactoryInterfaces('-interactive')
```

listJ2CActivationSpecs

Use the listJ2CActivationSpecs command to list the activation specifications that are contained under the resource adapter and message listener type that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-messageListenerType
Specifies the message listener type for the resource adapter for which you are making a list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CActivationSpecs $ra {-messageListener Type javax.jms.MessageListener}
```

- Using Jython string:

```
AdminTask.listJ2CActivationSpecs(ra, '[-messageListener Type javax.jms.MessageListener]')
```

- Using Jython list:

```
AdminTask.listJ2CActivation Specs(ra, ['-messageListener Type', 'javax.jms.MessageListener'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CActivationSpecs {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CActivationSpecs('-interactive')
```

listJ2CAdminObjects

Use the listJ2CAdminObjects command to list administrative objects that contain the administrative object interface that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-adminObjectInterface
Specifies the administrative object interface for which you want to list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CAdminObjects $ra {-adminObjectInterface
 fvt.adaptor.message.FVTMessageProvider}
```

- Using Jython string:

```
AdminTask.listJ2CAdminObjects(ra, '[-adminObjectInterface
 fvt.adaptor.message.FVTMessageProvider]')
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CAdminObjects {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CAdminObjects('-interactive')
```

listJ2CConnectionFactories

Use the listJ2CConnectionFactories command to list the Java 2 connector connection factories under the resource adapter and connection factory interface that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

-connectionFactoryInterface
Indicates the name of the connection factory that you want to list. This parameter is required.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listJ2CConnectionFactories $ra {-connectionFactoryInterface javax.sql.DataSource}
```

- Using Jython string:

```
AdminTask.listJ2CConnectionFactories(ra, '[-connectionFactoryInterface javax.sql.DataSource]')
```

- Using Jython list:

```
AdminTask.listJ2CConnectionFactories(ra, ['-connectionFactoryInterface',
 'javax.sql.DataSource'])
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listJ2CConnectionFactories {-interactive}
```

- Using Jython:

```
AdminTask.listJ2CConnectionFactories('-interactive')
```

listMessageListenerTypes

Use the listMessageListenerTypes command to list the message listener types that are defined under the resource adapter that you specify.

Target object

J2C Resource Adapter object ID

Parameters and return values

- Parameters: None
- Returns: A list of message listener types.

Examples

Batch mode example usage:

- Using Jacl:

```
$AdminTask listMessageListenerTypes $ra
```

- Using Jython string:

```
AdminTask.listMessageListenerTypes(ra)
```

- Using Jython list:

```
AdminTask.listMessageListenerTypes(ra)
```

Interactive mode example usage:

- Using Jacl:

```
$AdminTask listMessageListenerTypes {-interactive}
```

- Using Jython:

```
AdminTask.listMessageListenerTypes('-interactive')
```

Related tasks

"Configuring new J2C administrative objects using scripting" on page 1360
You can use scripting and the wsadmin tool to configure new J2C administrative objects.

"Configuring new J2C activation specifications using scripting" on page 1358
You can configure new J2C activation specifications using scripting and the wsadmin tool.

"Configuring new J2C resource adapters using scripting" on page 1354
Use the wsadmin tool to configure resource adapters with Resource Adapter Archive (RAR) files. A RAR file provides the classes and other code to support a resource adapter for access to a specific enterprise information system (EIS), such as the Customer Information Control System (CICS). Configure resource adapters for an EIS only after you install the appropriate RAR file.

Using the AdminTask object for scripted administration
Use the AdminTask object to access a set of administrative commands that provide an alternative way to access the configuration commands and the running object management commands.

Related reference

Commands for the AdminTask object
Use the AdminTask object to run administrative commands with the wsadmin tool.

Managing messaging with the WebSphere MQ messaging provider

Through this messaging provider, Java Message Service (JMS) messaging applications can use your WebSphere MQ system as an external provider of JMS messaging resources.

Before you begin

If your business uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominantly WebSphere MQ network, the WebSphere MQ messaging provider is the natural choice. However, there can be benefits in using another provider. If you are not sure which provider combination is best suited to your needs, see "Choosing messaging providers for a mixed environment" on page 1683.

About this task

The WebSphere MQ messaging provider supports JMS 1.1 domain-independent interfaces (sometimes referred to as "unified" or "common" interfaces). This enables applications to use the same interfaces for

both point-to-point and publish/subscribe messaging, and also enables both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, this approach is recommended for new applications. The domain-specific interfaces are supported for backwards compatibility for applications developed to use domain-specific queue interfaces, as described in section 1.5 of the JMS 1.1 specification.

The WebSphere MQ messaging provider also supports the J2EE Connector Architecture (JCA) 1.5 activation specification mechanism for message-driven beans (MDBs) across all platforms supported by WebSphere Application Server.

You can use WebSphere Application Server to configure WebSphere MQ resources for applications (for example queue connection factories) and to manage messages and subscriptions associated with JMS destinations. You administer security through WebSphere MQ.

The preferred solution for publish and subscribe messaging with WebSphere MQ as an external JMS messaging provider is to use a message broker such as WebSphere MQ Event Broker.

To use WebSphere MQ as a messaging provider for WebSphere Application Server, complete one or more of the following steps.

- Learn about interoperating with a WebSphere MQ network.
- Configure JMS resources for the WebSphere MQ messaging provider.

  You can do this through the WebSphere Application Server administrative console, or through the WebSphere Application Server set of WebSphere MQ administrative commands.
- List JMS resources for the WebSphere MQ messaging provider..

Related tasks

Learning about interoperating with a WebSphere MQ network

Configuring JMS resources for the WebSphere MQ messaging provider

Use the WebSphere Application Server administrative console to configure activation specifications, connection factories and destinations for the WebSphere MQ JMS provider.

Before you begin

This task assumes that you are working in a mixed WebSphere Application Server and WebSphere MQ environment, and that you have decided to use the WebSphere MQ messaging provider to handle JMS messaging between the two systems. If your business uses WebSphere MQ, and you want to integrate WebSphere Application Server messaging applications into a predominately WebSphere MQ network, the WebSphere MQ messaging provider is the natural choice. However, there can be benefits in using another provider. If you are not sure which provider combination is best suited to your needs, see “Choosing messaging providers for a mixed environment” on page 1683.

You can configure JMS resources for the WebSphere MQ messaging provider through the administrative console as described in this task, or you can configure JMS resources for the WebSphere MQ messaging provider through the WebSphere MQ administrative commands.

About this task

Note: The set of WebSphere Application Server administrative console panels used to configure JMS resources for the WebSphere MQ messaging provider has been redesigned to simplify the configuration process, and new panels have been introduced. A corresponding set of WebSphere MQ administrative commands has been introduced. There are new wizards for creating activation specifications and connection factories for the WebSphere MQ messaging provider and for converting listeners ports to activation specifications.

Using the administrative console, if you set the scope of the WebSphere MQ messaging provider to a scope which contains only WebSphere Application Server Version 6 or 7 nodes, you can configure JMS 1.1 resources and properties. This includes unified JMS connection factories for use by both point-to-point and publish/subscribe JMS 1.1 applications. With JMS 1.1, this approach is preferred to the domain-specific queue connection factory and topic connection factory.

If you set the scope of the WebSphere MQ messaging provider to a scope which contains only WebSphere Application Server Version 7 nodes, you can also configure JMS activations specifications.

If you set the scope to a WebSphere Application Server Version 5 node, you can only configure domain-specific JMS resources, and the subset of properties that apply to WebSphere Application Server Version 5.

For more information about configuring JMS resources for the WebSphere MQ messaging provider, see the following topics. These topics include optional steps for you to create a new JMS resource.

- "Creating an activation specification for the WebSphere MQ messaging provider using the Create Activation Specification wizard"
- "Configuring an existing activation specification for the WebSphere MQ messaging provider" on page 1854
- "Configuring a listener port to an activation specification for use with the WebSphere MQ messaging provider" on page 1855
- "Creating a connection factory for the WebSphere MQ messaging provider using the Create JMS Resource wizard" on page 1855
- "Configuring a JMS connection factory for the WebSphere MQ messaging provider" on page 1860
- "Configuring an existing JMS queue connection factory for the WebSphere MQ messaging provider" on page 1861
- "Configuring an existing JMS topic connection factory for the WebSphere MQ messaging provider" on page 1862
- "Configuring a JMS queue destination for the WebSphere MQ messaging provider" on page 1863
- "Configuring a JMS topic destination for the WebSphere MQ messaging provider" on page 1863

Creating an activation specification for the WebSphere MQ messaging provider using the Create Activation Specification wizard

Use this task create an activation specification for use with the WebSphere MQ messaging provider.

About this task

To modify an existing activation specification, see the related tasks.

To create an activation specification for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to create an activation specification.
2. Change the **Scope** setting to the level at which you want to create the activation specification.
3. Click **New** in the content pane to start the **Create WebSphere MQ activation specification** wizard.
   a. Select the WebSphere MQ messaging provider and click **OK**.
   b. In the **Configure basic attributes** panel, specify the following properties:

      Name The name by which this activation specification is known for administrative purposes within WebSphere Application Server.

      JNDI name
      The name that is used to bind this activation specification into the JNDI name space.

Description
Optional. A description of this activation specification for administrative purposes within WebSphere Application Server.

Click **Cancel** to cancel all actions and leave the wizard or click **Next** to continue.

4. In the **Select connection method** panel, specify how to connect to the WebSphere MQ messaging provider by selecting one of the following radio buttons:

   Enter all the required information into this wizard
   Click **Next** to proceed to the **Supply queue connection details** panel.

   Use a client channel definition table
   Click **Next** to proceed to the **Specify client channel definition table** panel.

5. In the **Specify client channel definition table** panel, enter information about the client channel definition table (CCDT) that is used to establish a connection to the WebSphere MQ messaging provider. Optionally, in the **Queue manager** field, enter the name of the queue manager to identify one or more entries to use from the CCDT. Click **Next** to continue to the **Test connection** panel.
6. In the **Supply queue connection details** panel, enter the name of the queue manager or queue sharing group that you want to connect to. Click **Next** to continue to the **Enter connection details** panel.
7. In the **Enter connection details** panel, specify the following properties:

   Transport
   Optional. The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ. For more information about configuring a transport type of *bindings then client* or *bindings*, refer to “Configuring the WebSphere MQ messaging provider with native libraries information” on page 1857and “Sizing the thread pools used by the WebSphere MQ messaging provider” on page 1857.

   Hostname
   The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

   Port Optional. The port number on which WebSphere MQ is listening.

   Server connection channel
   Optional. The WebSphere MQ server connection channel name used when connecting to WebSphere MQ queue manager or queue sharing group.

   Click **Cancel** to cancel all actions and leave the wizard or click **Next** to continue.

8. Optional: In the **Test connection** panel, if you want to test establishing the connection, click **Test connection**. This test can take several seconds to perform.
9. In the **Summary** panel, click **Cancel** to cancel all actions and leave the wizard, or **Previous** to return to the previous panel, or **Finish** to complete the creation of the new activation specification.
10. Stop then restart the application server.

Configuring an existing activation specification for the WebSphere MQ messaging provider

Use this task to browse or change an activation specification for use with the WebSphere MQ messaging provider.

About this task

To browse or change an activation specification for use with WebSphere MQ, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to view existing activation specifications, with a summary of their properties.
2. Change the **Scope** setting to the level at which the activation specification is visible to applications.

3. Click the name of the activation specification that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your activation specification.
7. Optional: Click **Broker properties** to display or change the list of broker properties of your activation specification.
8. Optional: Click **Custom properties** to display or change the list of custom properties of your activation specification.
9. Optional: Click **Client transport properties** to display or change the list of client transport properties of your activation specification. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
10. Click **OK**.
11. Save any changes to the master configuration.
12. To have the changed configuration take effect, stop then restart the application server.

Configuring a listener port to an activation specification for use with the WebSphere MQ messaging provider

Use this task to configure a listener port to an activation specification for use with the WebSphere MQ messaging provider.

About this task

Note: From WebSphere Application Server Version 7 listener ports are deprecated.

To migrate information from an deprecated listener port to an activation specification for use the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Servers** → **Application Servers** → *application_server*.
2. Under Communications, expand **Messaging** → **Message listener service** and click **Listener Ports** to view existing listener ports, with a summary of their properties.
3. Select the listener port that you want to work with by selecting the check box to its left.
4. Click **Convert to activation specification** to start the **Convert listener port to activation specification** wizard.
5. In the **Step1: Supply activation specification name** panel, enter the name of the new activation specification to be created, the JNDI name of the new activation specification, and the scope of the new activation specification (Server, Node, Cluster, Cell). Note that Cluster only appears when the server is in a cluster. Click **Next** to continue.
6. In the **Step2: Summary** panel, click **Finish** to complete the creation of the new activation specification.
7. Stop then restart the application server.
8. To complete the configuration of the activation specification, refer to Related tasks.

Creating a connection factory for the WebSphere MQ messaging provider using the Create JMS Resource wizard

Use this task create a connection factory, a queue connection factory, or a topic connection factory for use with the WebSphere MQ messaging provider.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories.

To modify an existing connection factory, see the related tasks.

To create a connection factory, a queue connection factory, or a topic connection factory for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories,** → **Queue connection factories or** → **Topic connection factories**. to create a connection factory, a queue connection factory or a topic connection factory.
2. Change the **Scope** setting to the level at which you want to create the connection factory.
3. Click **New** in the content pane to start the **Create JMS resource** wizard.
   a. Select the WebSphere MQ messaging provider and click **OK**.
   b. In the **Configure basic attributes** panel, specify the following properties:

      Name The name by which this connection factory is known for administrative purposes within WebSphere Application Server.

      JNDI name
      The name that is used to bind this connection factory into the JNDI name space.

      Description
      Optional. A description of this connection factory for administrative purposes within WebSphere Application Server.

      Click **Cancel** to cancel all actions and leave the wizard or click **Next** to continue.
4. In the **Select connection method** panel, specify how to connect to the WebSphere MQ messaging provider by selecting one of the following radio buttons:

   Enter all the required information into this wizard
   Click **Next** to proceed to the **Supply queue connection details** panel.

   Use a client channel definition table
   Click **Next** to proceed to the **Specify client channel definition table** panel.
5. In the **Specify client channel definition table** panel, enter information about the client channel definition table (CCDT) that is used to establish a connection to the WebSphere MQ messaging provider. Optionally, in the **Queue manager** field, enter the name of the queue manager to identify one or more entries to use from the CCDT. Click **Next** to continue to the **Test connection** panel.
6. In the **Supply queue connection details** panel, enter the name of the queue manager or queue sharing group that you want to connect to. Click **Next** to continue to the **Enter connection details** panel.
7. In the **Enter connection details** panel, specify the following properties:

   Transport
   Optional. The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ. For more information about configuring a transport type of *bindings then client* or *bindings*, refer to “Configuring the WebSphere MQ messaging provider with native libraries information” on page 1857and “Sizing the thread pools used by the WebSphere MQ messaging provider” on page 1857.

   Hostname
   The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

   Port Optional. The port number on which WebSphere MQ is listening.

   Server connection channel
   Optional. The WebSphere MQ server connection channel name used when connecting to WebSphere MQ queue manager or queue sharing group.

   Click **Cancel** to cancel all actions and leave the wizard or click **Next** to continue.

8. Optional: In the **Test connection** panel, if you want to test establishing the connection, click **Test connection**. This test can take several seconds to perform.
9. In the **Summary** panel, click **Cancel** to cancel all actions and leave the wizard, or **Previous** to return to the previous panel, or **Finish** to complete the creation of the new connection factory.
10. Stop then restart the application server.

Configuring the WebSphere MQ messaging provider with native libraries information:

To connect to a WebSphere MQ queue manager or queue sharing group in bindings mode, the WebSphere MQ messaging provider needs to know where to load native libraries from. This information is known as native path information. The way native path information is set depends on whether the connection is established in an application client or in an application server environment. Use this task to configure the WebSphere MQ messaging provider with native path information.

About this task

If you are running in a client environment, use launchClient to start a client application. In the system property `MQ_INSTALL_ROOT` enter the name of a directory which contains the WebSphere MQ native libraries, in a subdirectory of java/lib or java/lib64 depending on whether you are using 32 bit or 64 bit native libraries. For example, on Linux specify `./launchClient.sh myappclient.ear -CCDMQ_INSTALL_ROOT=/opt/mqm/.`

If you are running in an application server environment, you can configure the WebSphere MQ messaging provider with native path information using the command line, as described in "WMQAdminCommands command group for the AdminTask object" on page 1951, or you can use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. Select the WebSphere MQ messaging provider that is at the correct **Scope** for the connection factory or activation specification that will create the bindings mode connection. Note that native path information at `Server` scope is used in preference to native path information at higher scopes, and native path information at `Node` scope is used in preference to native path information at `Cell` scope
3. Under General Properties, in the **Native library path** property, enter the full name of the directory that contains the WebSphere MQ native libraries. For example, on Linux enter `/opt/mqm/java/lib`. Enter only one directory name.
4. Click **OK**.
5. Save any changes to the master configuration.
6. To have the changed configuration take effect, stop then restart the application server.

Sizing the thread pools used by the WebSphere MQ messaging provider:

Use this task to calculate the number of thread pools required by the WebSphere MQ messaging provider.

Before you begin

In IBM® WebSphere® Application Server, almost all of the work that is done by the WebSphere MQ messaging provider uses threads from the WMQCommonServices thread pool. By default, this thread pool has a maximum size of 40 threads in an application server environment and 10 threads in a client environment. If the number of threads is exceeded, work that is submitted by the WebSphere MQ messaging provider might not complete and errors might be output to the log files.

About this task

To prevent the thread pool from becoming exhausted, use the following guidelines to size the thread pool correctly for the application environment.

1. Application server environment If you are using a WebSphere MQ messaging provider connection factory, topic connection factory or queue connection factory, and all the following conditions apply:
   - The connection factory is enabled for two-phase commit
   - The transport mode is *bindings* or *bindings, then client*
   - The bindings leg is used

   then:
   - You require one thread for each connection. Allow enough threads to fill up the connection pool, which has a default maximum size of 10.
   - You require one thread for each session. Allow enough threads to fill up the session pool, which has a default maximum size of 10. There is one session pool for every connection in the connection pool.

   For example, each WebSphere MQ messaging provider connection factory with a *bindings* transport mode, and that is enabled for two-phase commit, requires 110 threads.

   Note: The previous information does not apply with WebSphere Application Server for z/OS®, even when you use a connection factory that is enabled for two-phase commit.

   If you use a WebSphere MQ messaging provider connection factory, topic connection factory or queue connection factory and both the following conditions apply:
   - The transport mode is *client* or *bindings, then client*
   - The client leg is used

   then:
   - You require one thread for each connection. Allow enough threads to fill up the connection pool, which has a default maximum size of 10.
   - You require one thread for each session. Allow enough threads to fill up the session pool, which has a default maximum size of 10. There is one session pool for every connection in the connection pool.

   For example, each WebSphere MQ messaging provider connection factory with a transport mode of *bindings, then client*, and that uses the client leg, requires 110 threads.

   If you use WebSphere MQ messaging provider activation specifications in either of the following circumstances:
   - To deliver messages to message-driven beans inside a transaction with any transport type
   - You are using a transport mode of *client* or *bindings, then client*, the client leg is used, and a transaction does not exist

   then:
   - You require one thread for each running message-driven bean (MDB) instance. The number of instances is determined by the maximum server sessions setting on a WebSphere MQ messaging provider activation specification. By default this value is 10.
   - You require one thread for each five MDB installations, if they all connect to the same queue manager. The default value for the WebSphere MQ messaging provider connectionConcurrency setting is 5. For more information on how to set the connectionConcurrency setting, refer to the Information Center for WebSphere MQ.
   - You require one extra thread to perform the dispatching of messages if the transport mode is *client* or *bindings, then client* and the client leg is used.

   For example, each WebSphere MQ messaging provider activation specification with a *client* transport mode, and that is enabled for two-phase commit, requires 12 threads.

   If you use message listener ports that use WebSphere MQ messaging provider resources to deliver messages to MDBs, you need to take account of the thread pool considerations for the connections and sessions that are used by the message listener port instances.

The message listener port uses one session for each server session. By default, there is one server session for each message listener port instance.

Then:

- You require one thread for each connection that is used by a message listener port instance. This thread is for the exception listener with which it is registered.
- You require one thread for each connection that is used by a message listener port instance. This thread is for the exception listener with which it is registered.
  - The message listener port is using a WebSphere MQ messaging provider connection factory that is enabled for two-phase commit
  - The transport mode is *bindings*, or *bindings, then client*
  - Uses the bindings leg

If the WebSphere MQ messaging provider connection factory has a transport mode of *client* or *bindings, then client*, and uses the client leg, one thread is required regardless of whether the connection factory is enabled for two-phase commit.

You require one thread for each message listener port instance for the dispatch thread that is used by the connection consumer object, if the message listener port conditions are true:

- Uses a WebSphere MQ messaging provider connection factory
- Has a transport mode of *client* or *bindings, then client*
- Uses the client leg

Note: For each installed MDB that is configured to use a particular message listener port, there is a single message listener port instance. For example, if three MDBs are configured to use the same message listener port, there are three message listener port instances.

If you install a single MDB using a message listener port which uses a WebSphere MQ messaging provider connection factory in *bindings* mode, and that is enabled for two-phase commit, you require at least four threads:

- One thread for the connection
- One thread for the session that is used inside the server session, which is used by the message listener port
- One thread for the exception listener
- One thread for the connection consumer

However, if the MDB is stopped and started, a different connection and therefore a different set of sessions might be retrieved from the pool. You need to allow 112 threads: 10 for the connection pool, 100 for the session pools, one for the exception listener, and one for the connection consumer.

2. Application server environment In the application client environment, the rules for thread requirements are similar to those rules in the application server environment. However, the rules are simplified because connection and session pooling is not used, and connection factories cannot take part inside a transaction. For most application clients, the default maximum thread pool size of 10 should be sufficient.

   If you use a WebSphere MQ messaging provider connection factory, topic connection factory or queue connection factory and both the following conditions apply:

   - The transport mode is *client* or *bindings, then client*
   - The client leg is used

   then:

   - You require one thread for each connection
   - You require one thread for each session

   If you are registering a message listener implementation with a session for asynchronous delivery of messages, you require an extra thread if the WebSphere MQ messaging provider connection factory that created the session has a transport mode of *client*, or *bindings, then client*, and the client leg is used.

If you register an exception listener implementation with a connection, you need an extra thread.

Consider the following scenario:

An IBM AIX® application server installation has a single MDB installed. The MDB has transactions managed by the Enterprise JavaBeans (EJB) container and has a transaction attribute of *required*, so messages are delivered to the MDB under an XA transaction. The MDB is configured to use a WebSphere MQ messaging provider activation specification with transport mode of *bindings.* When a message is received, the MDB performs some processing and then sends a reply message using WebSphere MQ messaging provider connection factory with a transport mode of *bindings*. The connection factory has the default connection and session pool settings.

The number of threads that are used by this scenario is 121 and consists of the following threads:

- 10 (one thread * default maximum number of MDB instances)
- 1 (the thread that is shared across deployments)
- 10 (the thread used by pooled connection * default pool size of 10)
- 100 (the thread used by pooled session * default connection pool size(10) * default session pool size (10))

3. Configuring WMQCommonServices thread pool size in an application server You need to configure every application server that uses WebSphere MQ messaging provider resources and the default WMQCommonServices thread pool size is not enough. For example, if the MDB that is used in the previous example is deployed in a clustered environment with two servers, perform the following procedure on both application servers to set the size of the WMQCommonServices thread pool to at least 121.

   To configure the maximum size of the WMQCommonServices thread pool in the application server, perform the following steps using the administrative console:

   a. Click **Servers** → **Server Types** → **WebSphere application servers***server name*.
   b. Under Additional Properties, click **Thread pools** → **WMQCommonServices** .
   c. Update the value of the **Maximum size** field to the relevant value and click **OK**.
   d. Click **Save** and restart the application server.

4. Configuring WMQCommonServices thread pool size in an application client To configure the maximum size of the WMQCommonServices thread pool in the application client container, complete the following steps:

   a. Create a file called *wmq.client.props*, and put it in the application client class path.
   b. Edit the *wmq.client.props* file and add the following text to the file:

      ```
      com.ibm.ws.wmqcsi.threadPoolMaximumSize=<desired size of thread pool>
      ```

      For example: `com.ibm.ws.wmqcsi.threadPoolMaximumSize=15`
   c. Save the changes.

Configuring a JMS connection factory for the WebSphere MQ messaging provider

Use this tasks to browse or configure an existing domain-independent connection factory for the WebSphere MQ messaging provider on an Application Server Version 7 node.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories. If you want to browse or configure a queue connection factory or topic connection factory, see the related tasks.

To create a connection factory, see the related tasks.

To browse or configure an existing connection factory for use with WebSphere MQ, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories** to view existing connection factories, with a summary of their properties.
2. Change the **Scope** setting to the level at which the connection factory is visible to applications.
3. Click the name of the connection factory that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your connection factory.
7. Optional: Click **Broker properties** to display or change the list of broker properties of your connection factory.
8. Optional: Click **Custom properties** to display or change the list of custom properties of your connection factory.
9. Optional: Click **Client transport properties** to display or change the list of client transport properties of your connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
10. Optional: Click **Connection pools** to display or change the connection pools detail of your connection factory.
11. Optional: Click **Session pools** to display or change the session pools detail of your connection factory.
12. Click **OK**.
13. Save any changes to the master configuration.
14. To have the changed configuration take effect, stop then restart the application server.

Configuring an existing JMS queue connection factory for the WebSphere MQ messaging provider

Use this task to browse or change an existing JMS queue connection factory for point-to-point messaging with the WebSphere MQ messaging provider.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories.

If you want to browse or change a connection factory, see the related tasks.

To browse or change a queue connection factory for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queue connection factories** to view existing queue connection factories, with a summary of their properties.
2. Change the **Scope** setting to the level at which the queue connection factory is visible to applications.
3. Click the name of the queue connection factory that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your queue connection factory.
7. Optional: Click **Custom properties** to display or change the list of custom properties of your queue connection factory.

8. Optional: Click **Client transport properties** to display or change the list of client transport properties of your queue connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
9. Optional: Click **Connection pools** to display or change the connection pools detail of your queue connection factory.
10. Optional: Click **Session pools** to display or change the session pools detail of your queue connection factory.
11. Click **OK**.
12. Save any changes to the master configuration.
13. To have the changed configuration take effect, stop then restart the application server.

Configuring an existing JMS topic connection factory for the WebSphere MQ messaging provider

Use this task to browse or change an existing JMS topic connection factory for publish/subscribe messaging with the WebSphere MQ messaging provider.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories. If you want to browse or configure a connection factory, see the related tasks.

To browse or configure a topic connection factory for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Topic connection factories** to view existing topic connection factories, with a summary of their properties.
2. Change the **Scope** setting to the level at which the topic connection factory is visible to applications.
3. Click the name of the topic connection factory that you want to work with.
4. Under **General properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your topic connection factory.
7. Optional: Click **Broker properties** to display or change the list of broker properties of your topic connection factory.
8. Optional: Click **Custom properties** to display or change the list of custom properties of your topic connection factory.
9. Optional: Click **Client transport properties** to display or change the list of client transport properties of your topic connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
10. Optional: Click **Connection pools** to display or change the connection pools detail of your topic connection factory.
11. Optional: Click **Session pools** to display or change the session pools detail of your topic connection factory.
12. Click **OK**.
13. Save any changes to the master configuration.
14. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS queue destination for the WebSphere MQ messaging provider

Use this task to browse or change a JMS queue destination for point-to-point messaging with the WebSphere MQ messaging provider. This task contains an optional step for you to create a new JMS queue destination.

About this task

To browse or change a queue destination for use with WebSphere MQ, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queues** to view existing queue destinations, with a summary of their properties.
2. Change the **Scope** setting to the level at which the queue destination is visible to applications.
3. To browse or change the properties of an existing queue destination, click its name in the list. Otherwise, to create a new queue, complete the following steps:
   a. Click **New** in the content pane. Select the **WebSphere MQ messaging provider** radio button and click **OK**.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

      Name The name by which this queue destination is known for administrative purposes within WebSphere Application Server.

      JNDI name
      The JNDI name that is used to bind the queue destination into the name space.

      Queue name
      The name of the WebSphere MQ queue to which messages are sent.
   c. Click **Apply**. This defines the queue destination to WebSphere Application Server, and enables you to browse or change additional properties.
4. Optional: Change WebSphere MQ queue settings, according to your needs.
5. Click **OK**.
6. Save any changes to the master configuration.
7. To have the changed configuration take effect, stop then restart the application server.

Configuring a JMS topic destination for the WebSphere MQ messaging provider

Use this task to browse or change a JMS topic destination for publish/subscribe messaging with the WebSphere MQ messaging provider. This task contains an optional step for you to create a new JMS topic destination.

About this task

To browse or change a topic destination for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Topics** to view existing topic destinations, with a summary of their properties.
2. Change the **Scope** setting to the level at which the topic destination is visible to applications.
3. To browse or change the properties of an existing topic destination, click its name in the list. Otherwise, to create a new topic destination, complete the following steps:
   a. Click **New** in the content pane. Select the **WebSphere MQ messaging provider** radio button and click **OK**.
   b. Specify the following required properties. You can specify other properties, as described in a later step.

Name The name by which this topic destination is known for administrative purposes within WebSphere Application Server.

JNDI name
The JNDI name that is used to bind the topic destination into the name space.

Topic name
The name of the WebSphere MQ topic to which messages are sent.

c. Click **Apply**. This defines the topic destination to WebSphere Application Server, and enables you to browse or change additional properties.

4. Optional: Change WebSphere MQ topics settings according to your needs.
5. Click **OK**.
6. Save any changes to the master configuration.
7. To have the changed configuration take effect, stop then restart the application server.

Configuring WebSphere MQ connection pooling

Browse or change properties of WebSphere MQ connection pooling for JMS connections from an application server to WebSphere MQ as a JMS provider.

About this task

Support for connection pooling does not affect the performance of a message listener, because it retains its connections while listening on a destination, but does affect the overall JMS system performance. When a connection is no longer required, WebSphere MQ can pool the connection then reuse it later instead of destroying it.

Note: This support is only available for use with WebSphere MQ as a JMS provider.

To enable WebSphere MQ connection pooling for an application server, use the administrative console to complete the following steps:

1. Display the Message Listener Service properties for the application server
   a. In the navigation pane, click **Servers** → **Application Servers**
   b. In the content pane, click the name of the application server.
   c. Under Additional Properties, click **Message Listener Service properties**.
2. Select Custom Properties, to enable WebSphere MQ connection pooling, add the following custom properties:
   mqjms.pooling.threshold
   The maximum number of unused connections in the pool.
   mqjms.pooling.timeout
   The timeout in milliseconds for unused connections in the pool.
3. Click **OK**.
4. Save any changes to the master configuration.
5. To have the changed configuration take effect, stop then restart the application server.

What to do next

For additional information, see the books in the WebSphere MQ library.

Configuring custom properties for the WebSphere MQ messaging provider

You can use WebSphere MQ as a messaging provider for WebSphere Application Server. Many of the properties for a WebSphere MQ messaging provider can be selected from panels in the WebSphere Application Server administration console, but there are further properties that you can configure as custom properties. Use this task to create custom properties for destinations and connection factories when using WebSphere MQ as a messaging provider.

About this task

To create a new custom property for a destination or for a connection factory when using WebSphere MQ as a messaging provider, use the administrative console to complete the following steps:

1. Display the WebSphere MQ messaging provider. In the navigation pane, expand **Resources** → **JMS**.
2. Select the type of JMS resource for which you want create custom properties.
3. Optional: Change the **Scope** setting to the level at which the resource definition is visible to applications.
4. In the contents pane, select the specific JMS resource name. This displays information about the resource.
5. To create custom properties, under Additional Properties, select **Custom properties** .
   a. Click **New** in the content pane.
   b. Specify the following properties.

      Name The name of the custom property. This property is required.

      Value The value for the custom property.

      Type The type of the custom property. Select the custom property type from the list.

   c. Click **Apply**. This defines the custom property to WebSphere Application Server, and enables you to browse or change additional properties.
6. Click **OK**.
7. Save any changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

WebSphere MQ custom properties:

WebSphere Application Server supports the use of custom properties to define WebSphere MQ properties. This is useful because it enables WebSphere Application Server to work with later versions of WebSphere MQ which might have properties that are not exposed in the WebSphere Application Server administrative console.

For instructions on how to create a new custom property, see “Configuring custom properties for the WebSphere MQ messaging provider” on page 1864.

In WebSphere Application Server Version 6.1, the custom properties that you define are validated by the WebSphere MQ client jar files contained in WebSphere Application Server. In previous versions, this was done within WebSphere Application Server itself, and then by the WebSphere MQ client jar files. If you have defined a property that is not valid for WebSphere MQ, the WebSphere MQ client jar files create an exception, which is caught by WebSphere Application Server, and logged in the Systemout.log and SystemErr.log. Examples of error messages are given at the end of this topic.

When a later version of WebSphere MQ is available that is supported by the WebSphere Application Server installation, new MQ properties might be created that are not known to WebSphere Application Server. You can configure these as custom properties through WebSphere Application Server so that they are recognized by the WebSphere MQ client jars. You can also configure WebSphere Application Server to point to the WebSphere MQ client jars in the external JMS provider, as described in “Configuring the WebSphere MQ messaging provider with native libraries information” on page 1857.

For information on valid values for WebSphere MQ properties, refer to *WebSphere MQ Using Java* or *WebSphere MQ System Administration*, which are available from the IBM Publications Center.

Error message example

The following example shows the type of text that the exception created by the client jars contains:

```
[09/02/06 15:40:06:377 GMT] 0000000a ContainerImpl E   WSVR0501E: Error creating
component null [class com.ibm.ws.runtime.component.ApplicationServerImpl]
com.ibm.ws.exception.RuntimeWarning: com.ibm.ws.runtime.component.binder.
ResourceBindingException: invalid configuration passed to resource binding logic.
REASON: Failed to create connection factory: Error raised constructing AdminObject,
error code: XAQCF PropertyName : XAQCF PropertyName
```

where `PropertyName` is the name of the invalid property.

Listing JMS resources for the WebSphere MQ messaging provider

Use the WebSphere Application Server administrative console to list JMS resources for the WebSphere MQ provider, for administrative purposes.

About this task

You use the WebSphere Application Server administrative console to list JMS resources, if you want to view, modify or delete any of the following resources:

- Connection factory (unified)
- Queue connection factory
- Topic connection factory
- Queue
- Topic
- Activation specification

When you use the Administrative Console to locate these resources, two different navigation pathways are available:

- Provider-centric navigation. This lets you view all providers (or just those for a specified scope if required), then navigate to a specific resource for a specific provider. This is the traditional way of navigating to a resource when you know which provider owns it. Any navigation that starts with **Resources** → **JMS** → **JMS providers** is provider-centric.
- Resource-centric navigation lets you view all resources of a specified type, then navigate to a resource. This is useful if you want to find a resource, but you do not know which provider owns it (you can list all resources of a given type across all scopes, for all providers, in a single panel). Any navigation that follows the pattern **Resources** → **JMS** → **[resource]**, where [resource] is one of the resource types listed above is resource-centric.

You can use either of these navigation pathways to locate resources of any type.

- Use provider-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**. This opens the providers collection which lists all currently configured providers across all scopes (you can modify the scope if required).
  3. From the providers collection, select the required provider. This opens the configuration tab for that provider. The configuration tab contains a set of links to all the resources owned by that provider.
  4. From the configuration tab, click the link for a resource type, for example the connection factories link. This opens the connection factories collection which lists all the connection factories for that provider.
  5. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties
- Use resource-centric navigation, for example to navigate to a specified connection factory
  1. Start the WebSphere Application Server administrative console.
  2. In the navigation pane, expand **Resources** → **JMS** → **Connection factories**. This opens the connection factories collection which lists all the connection factories across all providers.

3. From the connection factories collection, select the required connection factory. You can now view and work with the connection factory's properties.

WebSphere MQ messaging provider activation specification collection

Use this panel to select an activation specification to view or change its configuration properties. The activation specification is configured for the WebSphere MQ messaging provider for both point-to-point and publish/subscribe messaging.

This panel shows a list of the WebSphere MQ activation specifications with a summary of their configuration properties.

To view a list of WebSphere MQ activation specifications, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to display existing activation specifications.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the activation specifications are defined. This restricts the set of activation specifications displayed.
3. Optional: To define a new activation specification, click **New** to start the Create Activation Specification wizard.
4. Optional: To view or change the properties of an activation specification, click its name in the list displayed.

Configuring an existing activation specification for the WebSphere MQ messaging provider:

Use this task to browse or change an activation specification for use with the WebSphere MQ messaging provider.

About this task

To browse or change an activation specification for use with WebSphere MQ, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to view existing activation specifications, with a summary of their properties.
2. Change the **Scope** setting to the level at which the activation specification is visible to applications.
3. Click the name of the activation specification that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your activation specification.
7. Optional: Click **Broker properties** to display or change the list of broker properties of your activation specification.
8. Optional: Click **Custom properties** to display or change the list of custom properties of your activation specification.
9. Optional: Click **Client transport properties** to display or change the list of client transport properties of your activation specification. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
10. Click **OK**.
11. Save any changes to the master configuration.
12. To have the changed configuration take effect, stop then restart the application server.

WebSphere MQ messaging provider activation specification settings:

Use this panel to view or change the configuration properties of the selected activation specification for use with the WebSphere MQ messaging provider. These configuration properties control how connections are created to associated queues and topics.

To view WebSphere MQ activation specification settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to display existing activation specifications.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the activation specifications are defined. This restricts the set of activation specifications displayed.
3. Click the name of the activation specification that you want to work with.

Under General Properties there are four groups of properties:
- Administration
- Connection
- Destination
- Advanced

There are two variations of the **Connection** group:
- CCDT Connection settings group: A WebSphere MQ administrator can create a single Client Channel Definition Table (CCDT) of all the WebSphere MQ channels supported by queue managers in their enterprise. Additional WebSphere MQ configuration information can be associated with the entries in a CCDT. If the selected activation specification was created using a CCDT, only the following information is required to configure a connection:
  - Client channel definition table URL: The URL that specifies the location of the CCDT.
  - Queue manager: The queue manager name that specifies the CCDT entry, or entries, to use.
  - SSL configuration
- Explicitly-defined Connection settings group: If the selected activation specification was not created using a CCDT, the following information must be entered explicitly to configure a connection:
  - Queue manager
  - Transport
  - Hostname
  - Port
  - Server connection channel
  - Use SSL to secure communication with Websphere MQ: If you clear this property, the following properties cannot be used.
    - Centrally managed
    - Specific configuration
    - SSL configuration

Make any required changes to the Administration, Connection, Destination and Advanced groups of properties and then click **Apply** to save the configuration before, in the content pane under Additional Properties, you click any of the following links:
- **Advanced properties** to display or change the advanced properties of your WebSphere MQ activation specification
- **Broker properties** to display or change the broker properties of your WebSphere MQ activation specification
- **Custom properties** to display or change the custom properties of your WebSphere MQ activation specification

- **Client transport properties** to display or change the client transport properties of your WebSphere MQ activation specification. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the activation specification.

You can also specify the **localAddress** property using WebSphere MQ administrative commands. For more information about this property, refer to the “createWMQActivationSpec command” on page 1952.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

A WebSphere MQ activation specification has the following properties.

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Scope:

The scope specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the WebSphere MQ activation specification collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the activation specification is created.

For all activation specifications created using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which this activation specification is known for administrative purposes within WebSphere Application Server.

Data type String
Range The name must be unique within the set of activation specifications defined to the cell.

JNDI name:

The JNDI name that is used to bind the activation specification into the JNDI name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of this activation specification for administrative purposes within WebSphere Application Server.

Data type String

Client channel definition table URL:

A URL that points to a WebSphere MQ CCDT.

Data type String

Queue manager:

If this activation specification is based on a CCDT, this property is used to select an entry in the CCDT. Otherwise, it is the name of the queue manager or queue sharing group to connect to. A connection is established to this WebSphere MQ resource to receive messages.

Data type String
Range If this activation specification is based on a CCDT, the value must be one of the following:

- a valid queue manager name
- a valid queue manager name with the last character an asterisk
- an asterisk
- blank

If this connection factory is based on a CCDT, the value must be a valid queue manager name.

Transport:

The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ.

Data type Drop-down list
Default **bindings then client**

Range

client Use a TCP/IP based network connection to communicate with the WebSphere MQ queue manager.

bindings then client
Attempt a bindings mode connection to the queue manager. If this is not possible, revert to the client transport.

bindings
Establish a cross memory connection to a queue manager running on the same node. The following Client Transport Mode properties are disabled:
- Hostname
- Port
- Server connection channel

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to "Configuring the WebSphere MQ messaging provider with native libraries information" on page 1857and "Sizing the thread pools used by the WebSphere MQ messaging provider" on page 1857.

Hostname:

The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

Data type String

Port:

The port number on which WebSphere MQ is listening.

Data type Integer
Default 1414
Range The value must be in the range 1 to 65536 (inclusive).

Server connection channel:

The WebSphere MQ server connection channel name used when connecting to WebSphere MQ.

Data type String
Default SYSTEM.DEF.SVRCONN
Range The value must be a server connection channel defined to the WebSphere MQ queue manager being connected to.

Use SSL to secure communications with WebSphere MQ:

This option determines whether the SSL (Secure Sockets Layer) protocol is used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

When using a WebSphere MQ messaging provider activation specification in the application server environment, the application server manages SSL configuration. To change SSL configuration parameters, use the administrative console to navigate to the **Security** → **SSL certificate and key management** panel.

WebSphere MQ messaging provider activation specifications can only be configured with a single SSL cipher suite. If you specify more than one cipher suite in the SSL configuration for a WebSphere MQ messaging provider activation specification, only the first one is used.

Data type Checkbox. If this checkbox is cleared, the following SSL properties are disabled:

- Centrally managed
- Specific configuration
- SSL configuration

Centrally managed:

When the SSL protocol is used to communicate with WebSphere MQ, select this radio button to specify that the SSL configuration is taken from the centrally managed WebSphere Application Server SSL configuration.

When you select this radio button, the hostname and port attributes from the WebSphere MQ messaging provider activation specification are used to select an appropriate SSL configuration. To provide the SSL configuration which will be matched to the activation specification, see the Dynamic outbound endpoint SSL configuration settings topic listed under related reference.

Data type Radio button

Specific configuration:

Select this radio button when you want to specify a particular SSL configuration for use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

Data type Radio button

SSL configuration:

The specific SSL configuration to use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

This property is disabled if the **Centrally managed** radio button is selected and the WebSphere MQ messaging provider resource has been explicitly defined.

This property is always enabled if the WebSphere MQ messaging provider resource is based on a CCDT.

If this WebSphere MQ messaging provider resource is based on a CCDT, this parameter is only used if the relevant entries in the CCDT have been configured to use SSL.

Additionally, if a SSL configuration of `none` is selected, the default centrally managed WebSphere Application Server SSL configuration for the WebSphere MQ messaging provider is used.

Data type Drop-down list

Destination JNDI name:

The JNDI name for the JMS destination from which messages are consumed for delivery to an MDB that is configured to use this activation specification.

Data type String

Message selector:

A message selector expression specifying which messages are to be delivered.

Data type String

Destination type:

The type of destination (queue or topic) from which to consume messages.

Data type Drop-down list
Range
Queue The Destination JNDI name refers to a JMS destination that is a queue.
Topic The Destination JNDI name refers to a JMS destination that is a topic.

Durable subscription:

Whether a durable or nondurable subscription is used to deliver messages to an MDB subscribing to the topic.

Data type Checkbox
Default Cleared (nondurable)

Subscription name:

The name of a durable subscription. This is available only when the **Durable subscription** checkbox is selected.

Data type String

Authentication alias:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ.

Data type Drop-down list
Default (none)
Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ.

Client identifier:

The client identifier to specify when connecting to the WebSphere MQ messaging provider.

Data type String

Allow cloned durable subscriptions:

Determines whether multiple instances of a durable subscription can be accessed concurrently by different servers.

Data type Checkbox
Default Cleared
Range
Selected
Multiple instances of a durable subscription can be accessed concurrently by different servers.
Cleared
Multiple instances of a durable subscription can not be accessed concurrently by different servers.

Provider version:

The WebSphere MQ messaging provider version. This is used to determine whether to connect to a particular version of a queue manager. It is also used to determine the type of functionality required by the client.

Data type String
Range The value entered must be either the empty string or be must be in one of the following formats: n.n.n.n or n.n.n or n.n or n, where n is a numeric value greater than or equal to zero.

WebSphere MQ messaging provider activation specification advanced properties:

Use this panel to view or change the advanced properties of the selected activation specification for use with the WebSphere MQ messaging provider. These advanced properties control the behavior of connections made to WebSphere MQ messaging provider destinations.

To view WebSphere MQ activation specification advanced properties, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to display existing activation specifications.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the activation specifications are defined. This restricts the set of activation specifications displayed.
3. Click the name of the activation specification that you want to work with.
4. In the content pane, under Additional properties, click **Advanced properties** to view a list of the advanced properties of the WebSphere MQ activation specification.

Under General Properties there are four groups of properties:
- Message compression
- Connection consumer
- Message format
- Additional

Make any required changes to these groups and then click **Apply** to return to the activation specification.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ activation specification has the following advanced properties.

Compress message headers:

Enables the compression of message headers.

Data type Checkbox
Default Cleared
Range
Cleared
Do not compress message headers.
Selected
Compress message headers.

Compression algorithm for message payloads:

The compression algorithm to use to compress message payloads.

Data type Drop-down list
Default NONE
Range
RLE
ZLIBFAST
ZLIBHIGH
NONE

Retain messages, even if no matching consumer is available:

Determines whether messages for which there is no matching consumer are retained on the input queue or dealt with according with their disposition options.

Data type Checkbox
Default Selected
Range
Cleared
Do not retain messages.
Selected
Retain messages.

Rescan interval:

When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS) this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get messages from the WebSphere MQ queue. The setting determines how long a thread retrieves messages from a WebSphere MQ queue before switching to consume messages from another WebSphere MQ queue in the set, which is being asynchronously consumed from.

Data type Integer
Units Milliseconds
Default 5000
Range A value greater than zero.

Maximum server sessions:

The maximum number of server sessions in the server session pool used by the connection consumer.

Data type	Integer
Default	10
Range	A value greater than zero.

Server session pool timeout:

The period of time, in milliseconds, that an unused server session is held open in the server session pool before being closed due to inactivity.

Data type	Integer
Units	Milliseconds
Default	300,000
Range	A value greater than zero.

Start timeout:

The period of time, in milliseconds, within which delivery of a message to an MDB must start after the work to deliver the message has been scheduled. If this period of time elapses, the message is rolled back onto the queue.

Data type	Integer
Units	Milliseconds
Default	10,000
Range	A value greater than zero.

Coded character set identifier:

The character set to use when encoding strings in the message.

Data type	Integer
Default	819
Range	A value greater than zero. Must be one of the CCSIDs supported by WebSphere MQ.

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www.ibm.com/software/integration/wmq/library, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Fail JMS method calls if the WebSphere MQ queue manager is quiescing:

Selected JMS operations fail when the queue manager is put into a quiescing state. This enables the queue manager to quiesce successfully and shutdown.

Data type	Checkbox
Default	Selected

Range

Cleared
Do not fail JMS operations if the queue manager is quiescing.

Selected
Fail JMS operations if the queue manager is quiescing.

Stop endpoint if message delivery fails:

Determines whether message delivery is suspended to a failing endpoint.

Data type Checkbox
Default Selected
Range

Cleared
Message delivery is not suspended to a failing endpoint.

Selected
Message delivery is suspended to a failing endpoint when the **Number of sequential delivery failures before suspending endpoint** is exceeded.

Number of sequential delivery failures before suspending endpoint:

Sets the number of sequential message delivery failures to an endpoint that are allowed before message delivery to that endpoint is suspended. This property is enabled only when **Suspend message delivery to failing endpoints** is selected.

Data type Integer
Default 0
Range A value greater than or equal to zero.

WebSphere MQ messaging provider activation specification broker properties:

Use this panel to view or change the broker settings of the selected activation specification for use with the WebSphere MQ messaging provider. These broker settings determine how the WebSphere MQ messaging provider interacts with a broker for the purposes of publishing messages and subscribing to topics. Updates to the settings take effect when the server is restarted.

To view WebSphere MQ activation specification broker properties, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to display existing activation specifications.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the activation specifications are defined. This restricts the set of activation specifications displayed.
3. Click the name of the activation specification that you want to work with.
4. In the content pane under Additional properties, click **Broker properties** to view a list of the broker properties of the WebSphere MQ activation specification.

Under General Properties there are four groups of properties:
- Queues
- Capabilities
- Tuning
- Additional

Make any required changes to these groups and then click **Apply** to return to the activation specification.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ activation specification has the following broker properties.

Broker control queue:

The queue to which broker control messages are sent.

Data type	String
Default	SYSTEM.BROKER.CONTROL.QUEUE

Broker subscriber queue:

The queue from which subscription messages are received.

Data type	String
Default	SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

Broker connection consumer subscription queue:

The queue to receive subscription messages for connection consumers from.

Data type	String
Default	SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE

Broker connection consumer durable subscription queue:

The queue to receive subscription messages for durable connection consumers from.

Data type	String
Default	SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE

Version:

This determines some of the capabilities the broker is assumed to have. For example, whether to use a RFH version 1 or version 2 header in publications.

Data type	Radio button
Default	Version 1 broker
Range	**Version 1 broker** Message selection can not be specified. **Version 2 broker** Message selection can be specified. If you select this option, you must also complete **Specify where message selection occurs**.

Specify where message selection occurs:

This determines where message selection is performed. This property is enabled only if **Version 2 broker** was selected.

Data type Drop-down list
Default CLIENT
Range

CLIENT
Message selection is performed in the application server process.

BROKER
Message selection is performed in the broker process.

Subscription store:

This determines how subscriptions are tracked.

Data type Drop-down list
Default MIGRATE
Range

MIGRATE
Any information that is held using queues is migrated to the broker mechanism for persisting subscription information. If subscription information is already persisted using the broker mechanism then specifying a value of `Migrate` is equivalent to specifying a value of `Broker`.

BROKER
Internal broker mechanisms are used to track subscription information.

QUEUE
A designated WebSphere MQ queue is used to record information about current subscriptions.

Durable subscription state refresh interval:

This setting determines how often to recreate a long running transaction used to clean up durable subscriptions, for some versions of the queue manager.

Data type Integer
Default 60000
Range Any positive integer

Subscription cleanup level:

This setting determines how aggressively messages are cleaned up if the subscriber that is expected to consume the messages terminates unexpectedly.

Data type Drop-down list
Default SAFE

Range

SAFE A conservative algorithm is used to clean up subscriptions.

ASPROP
The cleanup algorithm is determined by a system property.

NONE No cleanup of subscriptions is performed.

STRONG
An aggressive algorithm is used to clean up subscriptions.

Subscription cleanup interval:

This setting determines how often to check for orphaned subscriptions and clean up messages.

Data type Integer
Default 3600000
Range Any positive integer

Subscription wildcard format:

This setting determines the wildcard format used for subscribing to more than one topic in a topic hierarchy.

Data type Drop-down list
Default character wildcards
Range

character wildcards
The following characters can stand for characters, or strings of characters in a topic name: '*' and '?'

'*' is interpreted as matching many characters

'?' is interpreted as matching a single character.

topic level wildcards
The following characters can stand for topics in a multilevel topic hierarchy: '+' and '#'

'+' is interpreted as matching a single topic name

'#' is interpreted as matching many topics in the hierarchy. '/' is used to delimit topics.

Optimize for sparse subscription patterns:

Specifies whether this activation specification is anticipated to receive a high proportion of messages that match its selection criteria. This information can be used to optimize message delivery.

Data type Checkbox
Default Cleared
Range

Cleared
Subscriptions frequently receive matching messages.

Selected
Subscriptions do not frequently receive matching messages.

Broker queue manager:

The name of the queue manager that is running the broker, if it is not the same as the queue manager to which the activation specification connects.

Data type	String
Default	The same queue manager name as specified in the activation specification.

WebSphere MQ messaging provider activation specification client transport properties:

Use this panel to view or change the client transport properties of an activation specification for use with the WebSphere MQ messaging provider. These properties affect how a client connection is established with a WebSphere MQ queue manager or queue sharing group. Updates to the properties take effect when the server is restarted.

To view WebSphere MQ activation specification client transport properties, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Activation specifications** to display existing activation specifications.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the activation specifications are defined. This restricts the set of activation specifications displayed.
3. Click the name of the activation specification that you want to work with.
4. In the content pane, under Additional properties, click **Client transport properties** to display a list of the client transport properties of the WebSphere MQ activation specification.

Under General Properties there are two groups of properties:

- Additional SSL settings
- Channel exits

Make any required changes to these groups and then click **Apply** to return to the activation specification.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ activation specification has the following client transport properties:

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Certificate revocation list:

A list of LDAP URLs pointing to LDAP repositories of SSL certificates that might have been revoked.

Data type	String
Default	No certificate revocation list
Range	The value must be a space-separated list of LDAP URLs.

Reset count:

The total number of bytes to transfer over an SSL connection before renegotiating the symmetric encryption keys used to secure the connection.

Data type	Integer
Default	0 (do not renegotiate)
Range	The value must be in the range 0 through 999,999,999 (inclusive).

Peer name:

A name (possibly including wildcards) that must match the distinguished name of the peer's SSL certificate for a connection to be established.

Data type	String
Default	Do not check the distinguished name of the peer's certificate.
Range	Validated using the rules for a WebSphere MQ SSLPEER channel parameter.

Receive exit or exits:

A comma-separated list of Java class names corresponding to receive exits to load.

Data type	String

Receive exit initialization data:

Initialization data to be passed to the receive exit.

Data type	String

Send exit or exits:

A comma-separated list of Java class names corresponding to send exits to load.

Data type	String

Send exit initialization data:

Initialization data to be passed to the send exit.

Data type	String

Security exit:

A Java class name corresponding to the security exit to load.

Data type	String

Security exit initialization data:

Initialization data to be passed to the security exit.

Data type String

WebSphere MQ connection factory collection

Use this task to select a connection factory to view or change its configuration properties. The unified JMS connection factory is configured for the WebSphere MQ messaging provider for both point-to-point and publish/subscribe messaging.

This panel shows a list of the WebSphere MQ connection factories with a summary of their configuration properties.

To view a list of the WebSphere MQ connection factories, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories** to display existing connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the connection factories are defined. This restricts the set of connection factories displayed.
3. Optional: To define a new connection factory, click **New** to start the Create Connection Factory wizard.
4. Optional: To view or change the properties of a connection factory, click its name in the list displayed.

Configuring a JMS connection factory for the WebSphere MQ messaging provider:

Use this tasks to browse or configure an existing domain-independent connection factory for the WebSphere MQ messaging provider on an Application Server Version 7 node.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories. If you want to browse or configure a queue connection factory or topic connection factory, see the related tasks.

To create a connection factory, see the related tasks.

To browse or configure an existing connection factory for use with WebSphere MQ, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories** to view existing connection factories, with a summary of their properties.
2. Change the **Scope** setting to the level at which the connection factory is visible to applications.
3. Click the name of the connection factory that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your connection factory.
7. Optional: Click **Broker properties** to display or change the list of broker properties of your connection factory.
8. Optional: Click **Custom properties** to display or change the list of custom properties of your connection factory.
9. Optional: Click **Client transport properties** to display or change the list of client transport properties of your connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.

10. Optional: Click **Connection pools** to display or change the connection pools detail of your connection factory.
11. Optional: Click **Session pools** to display or change the session pools detail of your connection factory.
12. Click **OK**.
13. Save any changes to the master configuration.
14. To have the changed configuration take effect, stop then restart the application server.

WebSphere MQ messaging provider connection factory settings:

Use this panel to view or change the configuration properties of the selected connection factory for use with the WebSphere MQ messaging provider. These configuration properties control how connections are created to associated JMS queues and topics.

The WebSphere MQ messaging provider supports JMS 1.1 domain-independent interfaces, such as the unified JMS connection factory. A domain-independent application can use the same interface for both point-to-point and publish/subscribe messaging, and can support both point-to-point and publish/subscribe messaging within the same transaction. With JMS 1.1, you are recommended to use domain-independent unified JMS connection factories for new applications. A domain-specific interface extends the domain-independent equivalent, so an application using domain-specific queue and topic connection factories can choose to use either interface.

To view WebSphere MQ connection factory settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories** to display existing connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the connection factories are defined. This restricts the set of connection factories displayed.
3. Click the name of the connection factory that you want to work with.

Under General Properties there are three groups of properties:
- Administration
- Connection
- Advanced

There are two variations of the **Connection** group:
- CCDT Connection settings group: A WebSphere MQ administrator can create a single Client Channel Definition Table (CCDT) of all the WebSphere MQ channels supported by queue managers in their enterprise. Additional WebSphere MQ configuration information can be associated with the entries in a CCDT. If the selected connection factory was created using a CCDT, only the following information is required to configure a connection:
  - Client channel definition table URL: The URL that specifies the location of the CCDT.
  - Queue manager: The queue manager name that specifies the CCDT entry, or entries, to use.
  - SSL configuration
- Explicitly-defined Connection settings group: If the selected connection factory was not created using a CCDT, the following information must be entered explicitly to configure a connection:
  - Queue manager
  - Transport
  - Hostname
  - Port
  - Server connection channel

– Use SSL to secure communication with Websphere MQ: If you clear this property, the following properties cannot be used.
  - Centrally managed
  - Specific configuration
  - SSL configuration

Make any required changes to the Administration, Connection and Advanced groups of properties and then click **Apply** to save the configuration before, in the content pane under Additional Properties, you click any of the following links:

- **Advanced properties** to display or change the advanced properties of your WebSphere MQ connection factory
- **Broker properties** to display or change the broker properties of your WebSphere MQ connection factory
- **Custom properties** to display or change the custom properties of your WebSphere MQ connection factory
- **Client transport properties** to display or change the client transport properties of your WebSphere MQ connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all the information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
- **Connection pools** to display or change the connection pools detail of your WebSphere MQ connection factory
- **Session pools** to display or change the session pools detail of your WebSphere MQ connection factory

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the connection factory.

You can specify the following additional properties using WebSphere MQ administrative commands:

- **localAddress**
- **clonedSubs**
- **containerAuthAlias**

For more information about these properties, refer to the “createWMQConnectionFactory command” on page 1966.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

A WebSphere MQ unified connection factory has the following properties:

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Scope:

Specifies the level at which this connection factory definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change other resources at a different scope, change the scope on the WebSphere MQ connection factory collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the queue connection factory is created.

For all connection factories using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which this connection factory is known for administrative purposes within WebSphere Application Server.

Data type String
Range The name must be unique within the set of connection factories defined to the cell.

JNDI name:

The JNDI name that is used to bind the connection factory into the JNDI name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of this connection factory for administrative purposes within WebSphere Application Server.

Data type String

Client channel definition table URL:

A URL that points to a WebSphere MQ CCDT.

Data type String

Queue manager:

If this connection factory is based on a CCDT, this property is used to select an entry in the CCDT. Otherwise, it is the name of the queue manager or queue sharing group to connect to. A connection is established to this WebSphere MQ resource to send or receive messages.

Data type String

Range If this connection factory is based on a CCDT, the value must be one of the following:

- a valid queue manager name
- a valid queue manager name with the last character an asterisk
- an asterisk
- blank

If this connection factory is not based on a CCDT, the value must be a valid queue manager name.

Transport:

The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ.

Data type Drop-down list

Default **bindings then client**

Range

client Use a TCP/IP based network connection to communicate with the WebSphere MQ queue manager

bindings then client
Attempt a bindings mode connection to the queue manager. If this is not possible, revert to the client transport.

bindings
Establish a cross memory connection to a queue manager running on the same node. The following Client Transport Mode properties are disabled:
- Hostname
- Port
- Server connection channel

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to “Configuring the WebSphere MQ messaging provider with native libraries information” on page 1857 and “Sizing the thread pools used by the WebSphere MQ messaging provider” on page 1857.

Hostname:

The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

Data type String

Port:

The port number on which WebSphere MQ is listening.

Data type	Integer
Default	1414
Range	The value must be in the range 1 to 65536 (inclusive).

Server connection channel:

The WebSphere MQ server connection channel name used when connecting to WebSphere MQ.

Data type	String
Default	SYSTEM.DEF.SVRCONN
Range	The value must be a server connection channel defined to the WebSphere MQ queue manager being connected to.

Use SSL to secure communications with WebSphere MQ:

This option determines whether the SSL (Secure Sockets Layer) protocol is used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

When using a WebSphere MQ messaging provider connection factory in the application server environment, the application server manages SSL configuration. To change SSL configuration parameters, use the administrative console to navigate to the **Security** → **SSL certificate and key management** panel.

When using a WebSphere MQ messaging provider connection factory in the client environment, the client takes SSL configuration information from the ssl.client.props file. Use of this file is detailed in the related reference information for this topic.

A WebSphere MQ messaging provider connection factory can only be configured with a single SSL cipher suite in the SSL configuration. If you specify more than one cipher suite in the SSL configuration for a WebSphere MQ messaging provider connection factory, only the first one is used.

Data type	Checkbox. If this checkbox is cleared, the following SSL properties are disabled: • Centrally managed • Specific configuration • SSL configuration

Centrally managed:

When the SSL protocol is used to communicate with WebSphere MQ, select this radio button to specify that the SSL configuration is taken from the centrally managed WebSphere Application Server SSL configuration.

When you select this radio button, the hostname and port attributes from the WebSphere MQ messaging provider connection factory are used to select an appropriate SSL configuration. To provide the SSL configuration which will be matched to the connection factory, see the Dynamic outbound endpoint SSL configuration settings topic listed under related reference.

Data type	Radio button

Specific configuration:

Select this radio button when you want to specify a particular SSL configuration for use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

Data type Drop-down list

SSL configuration:

The specific SSL configuration to use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

This property is disabled if the **Centrally managed** radio button is selected and the WebSphere MQ messaging provider resource has been explicitly defined.

This property is always enabled if the WebSphere MQ messaging provider resource is based on a CCDT.

If this WebSphere MQ messaging provider resource is based on a CCDT, this parameter is only used if the relevant entries in the CCDT have been configured to use SSL.

Additionally, if a SSL configuration of `none` is selected, the default centrally managed WebSphere Application Server SSL configuration for the WebSphere MQ messaging provider is used.

Data type Drop-down list

Component-managed authentication alias:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ.

Data type Drop-down list
Default (none)
Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ.

Client identifier:

The client identifier to specify when connecting to the WebSphere MQ messaging provider.

Data type String

Allow cloned durable subscriptions:

Determines whether multiple instances of a durable subscription can be accessed concurrently by different servers.

Data type Checkbox
Default Cleared
Range
Selected
Multiple instances of a durable subscription can be accessed concurrently by different servers.
Cleared
Multiple instances of a durable subscription can not be accessed concurrently by different servers.

Provider version:

The WebSphere MQ messaging provider version. This is used to determine whether to connect to a particular version of a queue manager. It is also used to determine the type of functionality required by the client.

Data type String

Range The value entered must be either the empty string or be must be in one of the following formats: n.n.n.n or n.n.n or n.n or n, where n is a numeric value greater than or equal to zero.

Support distributed two phase commit protocol:

Specifies whether the connection factory supports XA coordination of messaging transactions. Enable this option if multiple resources, including this connection factory, are to be used in the same transaction.

If you clear this property, you disable support for distributed two phase commit protocol. The JMS session can still be enlisted in a transaction, but it uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type Checkbox

Default Selected

Range

Selected
The connection factory supports the use of distributed two phase commit protocols for the coordination of transacted work.

Cleared
The connection factory does not support the use of distributed two phase commit protocols for the coordination of transacted work.

Recommendation Keep this option selected if transactions involve other resources, including other queues or topics. Clear this option only when you are certain that the queue manager connected to using this queue connection is the only resource in the transaction.

Authentication alias for XA recovery:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ during XA recovery.

Data type Drop-down list

Default (none)

Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ during XA recovery.

Mapping-configuration alias:

This field if used only in the absence of a login configuration on the component resource reference.

When the resource authority value is ″container″, the preferred way to define the authentication strategy is by specifying a login configuration and associated properties on the component resource reference.

If the **DefaultPrincipalMapping** login configuration is specified, the associated property will be a JAAS - J2C authentication data entry alias. To configure authentication information for use with the connection factory, under Related items, click **JAAS - J2C authentication data** .

Data type	Drop-down list
Default	(none)
Range	**ClientContainer** **WSLogin** **WSKRB5Login** **DefaultPrincipalMapping** **TrustedConnectionMapping** **KerberosMapping**

WebSphere MQ messaging provider connection factory advanced properties:

Use this panel to view or change the advanced properties of the selected connection factory for use with the WebSphere MQ messaging provider. These advanced properties control the behavior of connections made to WebSphere MQ messaging provider destinations.

To view WebSphere MQ connection factory advanced properties, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Connection factories** to display existing connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the connection factories are defined. This restricts the set of connection factories displayed.
3. Click the name of the connection factory that you want to work with.
4. In the content pane, under Additional properties, click **Advanced properties** to view a list of the advanced properties of the WebSphere MQ connection factory.

Under General Properties there are five groups of properties:

- Message compression
- Temporary destinations
- Connection consumer
- Message format
- Additional

Make any required changes to these groups and then click **Apply** to return to the connection factory.

Note:

- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ connection factory has the following advanced properties:

Compress message headers:

Enables the compression of message headers.

Data type	Checkbox
Default	Cleared

Range

Cleared
Do not compress message headers.
Selected
Compress message headers.

Compression algorithm for message payloads:

The compression algorithm to use to compress message payloads.

Data type Drop-down list
Default NONE
Range **RLE**
ZLIBFAST
ZLIBHIGH
NONE

WebSphere MQ model queue name:

The model queue that is used as a basis for temporary queue creation.

Data type String
Default SYSTEM.DEFAULT.MODEL.QUEUE

Temporary queue prefix:

The prefix to append to the beginning of the names generated for temporary queues.

Data type String

Temporary topic prefix:

The prefix to append to the beginning of the names generated for temporary topics.

Data type String

Retain messages, even if no matching consumer is available:

Determines whether messages for which there is no matching consumer are retained on the input queue or dealt with according with their disposition options.

Data type Checkbox
Default Selected
Range **Cleared**
Do not retain messages.
Selected
Retain messages.

Polling interval:

This setting is applicable in the client container only. When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS) this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get

messages from the WebSphere MQ queue. The setting determines how long a thread waits for a message to arrive at a WebSphere MQ queue before polling another WebSphere MQ queue in the set from which it is being asynchronously consumed.

Data type	Integer
Units	Milliseconds
Default	5000
Range	A value greater than zero.

Rescan interval:

When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS) this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get messages from the WebSphere MQ queue. The setting determines how long a thread retrieves messages from a WebSphere MQ queue before switching to consume messages from another WebSphere MQ queue in the set which is being asynchronously consumed from.

Data type	Integer
Units	Milliseconds
Default	5000
Range	A value greater than zero.

Maximum batch size:

The maximum number of messages to remove from a queue before at least one must be delivered to an asynchronous consumer.

Data type	Integer
Default	10
Range	A value greater than zero.

Coded character set identifier:

The character set to use when encoding strings in the message.

Data type	Integer
Default	819
Range	A value greater than zero. The coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www.ibm.com/software/integration/wmq/library, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Append an RFH version 2 header to reply messages:

The action to take when replying to a message without an RFH version 2 header.

Data type	Checkbox

Default
Selected

Range

Selected
Append an RFH version 2 header to reply messages.

Cleared
Do not append an RFH version 2 header to reply messages.

Fail JMS method calls if the WebSphere MQ queue manager is quiescing:

Selected JMS operations fail when the queue manager is put into a quiescing state. This enables the queue manager to quiesce successfully and shutdown.

Data type
Checkbox

Default
Selected

Range

Cleared
Do not fail JMS operations if the queue manager is quiescing.

Selected
Fail JMS operations if the queue manager is quiescing.

WebSphere MQ messaging provider connection factory broker properties:

Use this panel to view or change the broker settings of the selected connection factory or topic connection factory, for use with the WebSphere MQ messaging provider. These broker settings determine how the WebSphere MQ messaging provider interacts with a broker for the purposes of publishing messages and subscribing to topics. Updates to the settings take effect when the server is restarted.

To view WebSphere MQ connection factory or topic connection factory, broker properties, use the administrative console to complete the following steps:
1. In the navigation pane, either expand **Resources** → **JMS** → **Connection factories** to display existing connection factories or expand **Resources** → **JMS** → **Topic connection factories** to display existing topic connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the connection factories are defined. This restricts the set of connection factories displayed.
3. Click the name of the connection factory or topic connection factory that you want to work with.
4. In the content pane, under Additional Properties, click **Broker properties** to display the broker properties of the WebSphere MQ connection factory or topic connection factory.

Under General Properties there are four groups of properties:
- Queues
- Capabilities
- Tuning
- Additional

Make any required changes to these groups and then click **Apply** to return to the connection factory or topic connection factory.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ connection factory or topic connection factory has the following broker properties:

Broker control queue:

The queue to which broker control messages are sent.

Data type String
Default SYSTEM.BROKER.CONTROL.QUEUE

Broker publication queue:

The queue to which publication messages are sent.

Data type String
Default SYSTEM.BROKER.DEFAULT.STREAM

Broker subscriber queue:

The queue to which subscription messages are sent.

Data type String
Default SYSTEM.JMS.ND.SUBSCRIBER.QUEUE

Broker connection consumer subscription queue:

The queue to which subscription messages that are destined for a connection consumer are sent.

Data type String
Default SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE

Version:

This determines some of the capabilities the broker is assumed to have. For example, whether to use a RFH version 1 or version 2 header in publications.

Data type Radio button
Default Version 1 broker
Range

Version 1 broker
Message selection can not be specified.

Version 2 broker
Message selection can be specified. If you select this option, you must also complete **Specify where message selection occurs**.

Specify where message selection occurs:

This determines where message selection is performed. This property is enabled only if **Version 2 broker** was selected.

Data type Drop-down list
Default CLIENT
Range

CLIENT
Message selection is performed in the application server process.

BROKER
Message selection is performed in the broker process.

Subscription store:

This setting determines how subscriptions are tracked.

Data type Drop-down list
Default MIGRATE
Range

MIGRATE
Any information that is held using queues is migrated to the broker mechanism for persisting subscription information. If subscription information is already persisted using the broker mechanism then specifying a value of `Migrate` is equivalent to specifying a value of `Broker`.

BROKER
Internal broker mechanisms are used to track subscription information.

QUEUE
A designated WebSphere MQ queue is used to record information about current subscriptions.

Durable subscription state refresh interval:

This setting determines how often to recreate a long running transaction used to clean up durable subscriptions, for some versions of the queue manager.

Data type Integer
Default 60000
Range Any positive integer

Subscription cleanup level:

This setting determines how aggressively messages are cleaned up if the subscriber that is expected to consume the messages terminates unexpectedly.

Data type Drop-down list
Default SAFE

Range

SAFE A conservative algorithm is used to clean up subscriptions.

ASPROP
The cleanup algorithm is determined by a system property.

NONE No cleanup of subscriptions is performed.

STRONG
An aggressive algorithm is used to clean up subscriptions.

Subscription cleanup interval:

This setting determines how often to check for orphaned subscriptions and clean up messages.

Data type Integer
Default 3600000
Range Any positive integer

Subscription wildcard format:

This setting determines the wildcard format used for subscribing to more than one topic in a topic hierarchy.

Data type Drop-down list
Default character wildcards
Range

character wildcards
The following characters can stand for characters, or strings of characters in a topic name: '*' and '?'

'*' is interpreted as matching many characters

'?' is interpreted as matching a single character.

topic level wildcards
The following characters can stand for topics in a multilevel topic hierarchy: '+' and '#'

'+' is interpreted as matching a single topic name

'#' is interpreted as matching many topics in the hierarchy. '/' is used to delimit topics.

Publish acknowledgement window:

Specifies the number of messages to publish before publishing a message which requires broker acknowledgement

Data type Integer
Default 25
Range Any positive integer

Optimize for sparse subscription patterns:

Specifies whether this connection factory is anticipated to receive a high proportion of messages that match its selection criteria. This information can be used to optimize message delivery.

Data type Checkbox
Default Cleared
Range
Cleared
Subscriptions frequently receive matching messages.
Selected
Subscriptions do not frequently receive matching messages.

Broker queue manager:

The name of the queue manager that is running the broker, if it is not the same as the queue manager to which the connection factory connects.

Data type String
Default The same queue manager name as specified in the connection factory.

WebSphere MQ resource custom properties settings:

Use this page to specify custom properties that your enterprise information system (EIS) requires for the resource providers and resource factories that you configure. For example, most database vendors require additional custom properties for data sources that access the database.

To view this administrative console page, complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If **Scope** is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, click the **WebSphere MQ messaging provider** that you want to support the JMS destination.
4. In the content pane, under Additional Properties, click the type of resource that you want to change, for example **Queues**.
5. Click the name of the resource that you want to work with.
6. In the content pane, under General Properties, complete the groups of fields, for example **Administration** and **WebSphere MQ Queue**.
7. In the content pane, under Additional Properties, click **Custom properties** to display a list of the custom properties of a WebSphere MQ resource.

A resource for use with the WebSphere MQ messaging provider has the following custom properties.

Note:
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Name:

The name by which the resource is known for administrative purposes within IBM WebSphere Application Server.

Data type String

Value:

The value of the resource.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

This name is used to link the platform binding information. The binding associates the resources defined by the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.

Data type String

Description:

A description of the resource, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

Required:

Whether or not the resource is required, for administrative purposes within IBM WebSphere Application Server.

Data type String
Default Null

WebSphere MQ messaging provider connection factory client transport settings:

Use this panel to view or change the client transport settings of a connection factory, queue connection factory or topic connection factory for use with the WebSphere MQ messaging provider. Client transport properties affect how a client connection is established with a WebSphere MQ queue manager or queue sharing group. Updates to the settings take effect when the server is restarted.

To view WebSphere MQ connection factory, queue connection factory or topic connection factory, client transport settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS**.
2. Click **Connection factories**, **Queue connection factories** or **Topic connection factories** to display existing connection factories, queue connection factories or topic connection factories.
3. Click the name of the connection factory, queue connection factory or topic connection factory that you want to work with.
4. In the content pane under Additional Properties, click **Client transport properties** to view a list of the client transport settings of the WebSphere MQ connection factory, queue connection factory or topic connection factory.

Under General Properties there are two groups of properties:
- Additional SSL settings
- Channel exits

Make any required changes to these groups and then click **Apply** to return to the connection factory, queue connection factory or topic connection factory.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ connection factory, queue connection factory or topic connection factory has the following client transport settings properties:

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Certificate revocation list:

A list of LDAP URLs pointing to LDAP repositories of SSL certificates that might have been revoked.

Data type	String
Default	No certificate revocation list
Range	The value must be a space-separated list of LDAP URLs.

Peer name:

A name (possibly including wildcards) that must match the distinguished name of the peer's SSL certificate for a connection to be established.

Data type	String
Default	Do not check the distinguished name of the peer's certificate.
Range	Validated using the rules for a WebSphere MQ SSLPEER channel parameter.

Reset count:

The total number of bytes to transfer over an SSL connection before renegotiating the symmetric encryption keys used to secure the connection.

Data type	Integer
Default	0 (do not renegotiate)
Range	The value must be in the range 0 through 999,999,999 (inclusive).

Receive exits:

A comma-separated list of Java class names corresponding to receive exits to load.

Data type	String

Receive exit initialization data:

Initialization data to be passed to the receive exit.

Data type String

Send exits:

A comma-separated list of Java class names corresponding to send exits to load.

Data type String

Send exit initialization data:

Initialization data to be passed to the send exit.

Data type String

Security exit:

A Java class name corresponding to the security exit to load.

Data type String

Security exit initialization data:

Initialization data to be passed to the security exit.

Data type String

Connection pool settings:

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types, like JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

For example: click **Resources** → **JDBC** → **JDBC Providers** → ***JDBC_provider*** → **Data Sources** → ***data_source*** → **Connection pool properties**

The path for JMS queue connection factories is:**Resources** → **JMS** → **Queue connection factories** → ***JMS_queue_connection_factory*** → **[Additional properties] Connection pool properties**.

Connection timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds that a connection request waits when there are no connections available in the free pool and no new connections can be created. This usually occurs because the maximum value of connections in the particular connection pool has been reached.

For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. In most cases you should not retry the getConnection() method; if a longer wait time is required you should increase the Connection Timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.

If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Maximum connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown. For example: If the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

Knowing the number of connection pools that can potentially request connections from the backend (such as a DB2 database or a CICS server) helps you determine a value for the Maximum Connections property.

For multiple standalone application servers that use the same data source configuration, or J2C connection factory configuration, a separate physical connection pool exists for each server. If you clone these same application servers, WebSphere Application Server implements a separate connection pool for each clone.

All of these connection pools correspond to the same data source or connection factory configuration. Therefore all of these connection pools can potentially request connections from the same backend resource, at the same time. The single Maximum Connections value that you set on this console panel applies to every one of these connection pools. Consequently, setting a high Maximum Connections value can result in a load of connection requests that overwhelms your backend resource.

Data type	Integer
Default	10
Range	0 to maximum integer If Max Connections is set to 0, the Connection Timeout value is ignored.

Note: For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum connections:

Specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example, if the Minimum Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting Aged Timeout to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached, the application server will not discard the connection until after the transaction is completed and the connection is closed.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**.

Data type	String

Defaults

- EntirePool for J2C connection factories and JMS-related connection factories
- EntirePool for WebSphere Version 4.0 data sources
- EntirePool for current version data sources that you create through the administrative console
- EntirePool for current version data sources that you script through wsadmin AdminConfig commands, invoking JDBC templates that are built into WebSphere Application Server (For information on the command **createUsingTemplate**, see the information center article ″Commands for the AdminConfig object.″)
- FailingConnectionOnly for data sources that you script in wsadmin without invoking JDBC templates

:

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

Session pool settings:

Use this page to configure session pool settings.

This administrative console page is common to a range of resource types; for example, JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select

an instance of the resource provider, then an instance of the resource type, then click **Session pools**. For example: click **Resources** → **JMS** → **JMS providers**-*>default messaging provider*-**>Queue connection factories**-*>connection_factory*-**>Session pools**.

Connection Timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

The wait is necessary when the maximum value of connections (**Max Connections**) to a particular connection pool is reached . For example, if *Connection Timeout* is set to 300 and the maximum number of connections is reached, the Pool Manager waits for 300 seconds for an available physical connection. If a physical connection is *not* available within this time, the Pool Manager throws a ConnectionWaitTimeoutException. It usually does not make sense to retry the getConnection() method, because if a longer wait time is required, you should set the **Connection Timeout** setting to a higher value. Therefore, if this exception is caught by the application, the administrator should review the expected usage of the application and tune the connection pool and the database accordingly.

If Connection Timeout is set to 0, the Pool Manager waits as long as necessary until a connection is allocated (which happens when the number of connections falls below the value of **Max Connections**).

If Max Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Max Connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown.

For example, if the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

If Max Connections is set to 0, the Connection Timeout value is ignored.

For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

If clones are used, one data pool exists for each clone. Knowing the number of data pools is important when configuring the database maximum connections.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Data type	Integer
Default	10
Range	0 to max int

Min Connections:

Specifies the minimum number of physical connections to maintain.

Until this number is reached, the pool maintenance thread does not discard physical connections. However, no attempt is made to bring the number of connections up to this number. If you set a value for Aged Timeout, the minimum is not maintained. All connections with an expired age are discarded.

For example if the **Min Connections** value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the **Min Connections** setting.

Data type	Integer
Default	1
Range	0 to max int

Reap Time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if **Reap Time** is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the **Unused Timeout** and **Aged Timeout** settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in **Min Connections**. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused Timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections not in use exceeds the **Min Connections** setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is

discarded. Note that accuracy of this timeout, as well as performance, is affected by the **Reap Time** value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged Timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting **Aged Timeout** to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the **Reap Timeout** value for optimal performance. For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. Note that accuracy of this timeout, as well as performance, are affected by the Reap Time value. For more information, see Reap Time.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge Policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**. Java EE Connector Architecture (JCA) data sources can have either option. WebSphere Version 4.0 data sources always have a purge policy of **EntirePool**.

Data type	String
Default	FailingConnectionOnly

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and throws a *StaleConnectionException* during the next operation on that connection. Subsequent *getConnection* requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the *StaleConnectionException* is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next *getConnection* request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

WebSphere MQ queue connection factory collection

Use this panel to select a queue connection factory to view or change its configuration properties. The queue connection factory is configured for the WebSphere MQ messaging provider for point-to-point messaging with JMS queues.

This panel shows a list of WebSphere MQ queue connection factories with a summary of their configuration properties.

To view the WebSphere MQ queue connection factory collection, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queue connection factories** to display existing queue connection factories.
2. If appropriate, in the content pane, change the **Scope** to restrict the set of queue connection factories displayed. If **Scope** is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. Optional: To define a new queue connection factory, click **New**.
4. Optional: Optional: To view or change the properties of a queue connection factory, click its name in the list displayed.

Configuring an existing JMS queue connection factory for the WebSphere MQ messaging provider:

Use this task to browse or change an existing JMS queue connection factory for point-to-point messaging with the WebSphere MQ messaging provider.

About this task

With JMS 1.1, domain-independent connection factories are preferred to domain-specific queue connection factories and topic connection factories.

If you want to browse or change a connection factory, see the related tasks.

To browse or change a queue connection factory for use with the WebSphere MQ messaging provider, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queue connection factories** to view existing queue connection factories, with a summary of their properties.
2. Change the **Scope** setting to the level at which the queue connection factory is visible to applications.
3. Click the name of the queue connection factory that you want to work with.
4. Under **General Properties** make modifications as necessary. Use the administrative console help for information about each of the available fields.
5. Click **Apply** to save the configuration.
6. Optional: Click **Advanced properties** to display or change the list of advanced properties of your queue connection factory.
7. Optional: Click **Custom properties** to display or change the list of custom properties of your queue connection factory.
8. Optional: Click **Client transport properties** to display or change the list of client transport properties of your queue connection factory. This link is only present on the explicitly-defined variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
9. Optional: Click **Connection pools** to display or change the connection pools detail of your queue connection factory.
10. Optional: Click **Session pools** to display or change the session pools detail of your queue connection factory.
11. Click **OK**.
12. Save any changes to the master configuration.
13. To have the changed configuration take effect, stop then restart the application server.

WebSphere MQ messaging provider queue connection factory settings:

Use this panel to view or change the configuration properties of the selected queue connection factory for use with the WebSphere MQ messaging provider. These configuration properties control how connections are created to associated JMS queue destinations.

A WebSphere MQ queue connection factory is used to create JMS connections to queues provided by WebSphere MQ for point-to-point messaging.

To view WebSphere MQ queue connection factory settings, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queue connection factories** to display existing queue connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the queue connection factories are defined. This restricts the set of queue connection factories displayed.
3. Click the name of the queue connection factory that you want to work with.

Under General Properties there are three groups of properties:

- Administration
- Connection

- Advanced

There are two variations of the **Connection** group:

- CCDT Connection settings group: A WebSphere MQ administrator can create a single Client Channel Definition Table (CCDT) of all the WebSphere MQ channels supported by queue managers in their enterprise. Additional WebSphere MQ configuration information can be associated with the entries in a CCDT. If the selected connection factory was created using a CCDT, only the following information is required to configure a connection:
  - Client channel definition table URL: The URL that specifies the location of the CCDT.
  - Queue manager: The queue manager name that specifies the CCDT entry, or entries, to use.
  - SSL configuration
- Explicitly-defined Connection settings group: If the selected connection factory was not created using a CCDT, the following information must be entered explicitly to configure a connection:
  - Queue manager
  - Transport
  - Hostname
  - Port
  - Server connection channel
  - Use SSL to secure communication with Websphere MQ: If you clear this property, the following properties do not appear on the panel.
    - Centrally managed
    - Specific configuration
    - SSL configuration

Make any required changes to the Administration, Connection and Advanced groups of properties and then click **Apply** before, in the content pane under Additional Properties, you click any of the following links:

- **Advanced properties** to display the advanced properties of your WebSphere MQ queue connection factory
- **Custom properties** to display the custom properties of your WebSphere MQ queue connection factory
- **Client transport properties** to display the client transport properties of your WebSphere MQ queue connection factory. This link is only present on the explicitly-specified variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
- **Connection pools** to display the connection pool detail of your WebSphere MQ queue connection factory
- **Session pools** to display the session pools detail of your WebSphere MQ queue connection factory

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the connection factory.

You can specify the following additional properties using WebSphere MQ administrative commands:

- **localAddress**
- **containerAuthAlias**

For more information about these properties, refer to the “createWMQConnectionFactory command” on page 1966.

Note: When specifying WebSphere MQ properties, the following restrictions apply:

- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

A WebSphere MQ queue connection factory has the following properties:

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource (or other resources) at a different scope, change the scope on the WebSphere MQ queue connection factory collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the queue connection factory is created.

For all queue connection factories created using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which this queue connection factory is known for administrative purposes within WebSphere Application Server.

Data type String
Range The name must be unique within the set of queue connection factories defined to the cell.

JNDI name:

The JNDI name that is used to bind the queue connection factory into the name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of this queue connection factory for administrative purposes within WebSphere Application Server.

Data type	String
Default	Null

Client channel definition table URL:

A URL that points to a WebSphere MQ CCDT.

Data type	String

Queue manager:

If this queue connection factory is based on a CCDT, this property is used to select an entry in the CCDT. Otherwise, it is the name of the queue manager or queue sharing group to connect to. A connection is established to this WebSphere MQ resource to send or receive messages.

Data type	String
Range	If this queue connection factory is based on a CCDT, the value must be one of the following: • a valid queue manager name • a valid queue manager name with the last character an asterisk • an asterisk • blank If this connection factory is not based on a CCDT, the value must be a valid queue manager name.

Transport:

The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ.

Data type	Drop-down list
Default	**bindings then client**

Range

client Use a TCP/IP based network connection to communicate with the WebSphere MQ queue manager

bindings then client
Attempt a bindings mode connection to the queue manager. If this is not possible, revert to the client transport.

bindings
Establish a cross memory connection to a queue manager running on the same node. The following Client Transport Mode properties are disabled:
- Hostname
- Port
- Server connection channel

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to "Configuring the WebSphere MQ messaging provider with native libraries information" on page 1857 and "Sizing the thread pools used by the WebSphere MQ messaging provider" on page 1857.

Hostname:

The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

Data type String

Port:

The port number on which WebSphere MQ is listening.

Data type Integer
Default 1414
Range The value must be in the range 1 to 65536 (inclusive).

Server connection channel:

The WebSphere MQ server connection channel name used when connecting to WebSphere MQ.

Data type String
Default SYSTEM.DEF.SVRCONN
Range The value must be a server connection channel defined to the WebSphere MQ queue manager being connected to.

Use SSL to secure communications with WebSphere MQ:

This option determines whether the SSL (Secure Sockets Layer) protocol is used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

When using a WebSphere MQ messaging provider queue connection factory in the application server environment, the application server manages SSL configuration. To change SSL configuration parameters, use the administrative console to navigate to the **Security** → **SSL certificate and key management** panel.

When using a WebSphere MQ messaging provider queue connection factory in the client environment, the client takes SSL configuration information from the ssl.client.props file. Use of this file is detailed in the related reference information for this topic.

You can only use one cipher suite in the SSL configuration for a WebSphere MQ messaging provider queue connection factory. If you specify more than one cipher suite, only the first one is used.

Data type Checkbox. If this checkbox is cleared, the following SSL properties are disabled:

- Centrally managed
- Specific configuration
- SSL configuration

Centrally managed:

When the SSL protocol is used to communicate with WebSphere MQ, select this radio button to specify that the SSL configuration is taken from the centrally managed WebSphere Application Server SSL configuration.

When you select this radio button, the hostname and port attributes from the WebSphere MQ messaging provider queue connection factory are used to select an appropriate SSL configuration. To provide the SSL configuration which will be matched to the queue connection factory, see the Dynamic outbound endpoint SSL configuration settings topic listed under related reference.

Data type Radio button

Specific configuration:

Select this radio button when you want to specify a particular SSL configuration for use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

Data type Radio button

SSL configuration:

The specific SSL configuration to use when SSL is to be used to secure network communications with the WebSphere queue manager or queue sharing group.

This property is disabled if the **Centrally managed** radio button is selected and the WebSphere MQ messaging provider resource has been explicitly defined.

This property is always enabled if the WebSphere MQ messaging provider resource is based on a CCDT.

If this WebSphere MQ messaging provider resource is based on a CCDT, this parameter is only used if the relevant entries in the CCDT have been configured to use SSL.

Additionally, if a SSL configuration of `none` is selected, the default centrally managed WebSphere Application Server SSL configuration for the WebSphere MQ messaging provider is used.

Data type Drop-down list

Component-managed authentication alias:

The authentication alias which specifies the user name and password to use when connecting to the WebSphere MQ messaging provider.

Data type Drop-down list
Default (none)
Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ.

Client identifier:

The client identifier to specify when connecting to WebSphere MQ messaging provider.

Data type String

Provider version:

The WebSphere MQ messaging provider version. This is used to determine whether to connect to a particular version of a queue manager. It is also used to determine the type of functionality required by the client.

Data type String
Range The value entered must be either the empty string or be must be in one of the following formats: n.n.n.n or n.n.n or n.n or n, where n is a numeric value greater than or equal to zero.

Support distributed two phase commit protocol:

Specifies whether the connection factory supports XA coordination of messaging transactions. Enable this option if multiple resources, including this connection factory, are to be used in the same transaction.

If you clear this property, you disable support for distributed two phase commit protocol. The JMS session can still be enlisted in a transaction, but it uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type Checkbox
Default Selected
Range
Selected
The queue connection factory supports the use of distributed two phase commit protocols for the coordination of transacted work.
Cleared
The queue connection factory does not support the use of distributed two phase commit protocols for the coordination of transacted work.
Recommended Keep this option selected if transactions involve other resources, including other queues or topics. Clear this option only when you are certain that the queue manager connected to using this queue connection factory is the only resource in the transaction.

Authentication alias for XA recovery:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ during XA recovery.

Data type	Drop-down list
Default	(none)
Range	All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ during XA recovery.

Mapping-configuration alias:

This field if used only in the absence of a login configuration on the component resource reference.

When the resource authority value is ″container″, the preferred way to define the authentication strategy is by specifying a login configuration and associated properties on the component resource reference.

If the **DefaultPrincipalMapping** login configuration is specified, the associated property will be a JAAS - J2C authentication data entry alias. To configure authentication information for use with the connection factory, under Related items, click **JAAS - J2C authentication data** .

Data type	Drop-down list
Default	(none)
Range	**ClientContainer** **WSLogin** **WSKRB5Login** **DefaultPrincipalMapping** **TrustedConnectionMapping** **KerberosMapping**

Container-managed authentication alias:

The authentication alias which specifies the user name and password to use when connecting to the WebSphere MQ messaging provider.

Data type	Drop-down list
Default	(none)
Range	All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ.

WebSphere MQ messaging provider queue connection factory advanced properties:

Use this panel to view or change the advanced properties of the selected queue connection factory for use with the WebSphere MQ messaging provider. These advanced properties control the behavior of connections made to WebSphere MQ messaging provider destinations.

To view WebSphere MQ queue connection factory advanced properties, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Queue connection factories** to display existing queue connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the queue connection factories are defined. This restricts the set of queue connection factories displayed.

3. Click the name of the queue connection factory that you want to work with.
4. In the content pane, under Additional properties, click **Advanced properties** to view a list of the advanced properties of the WebSphere MQ queue connection factory.

Under General Properties there are five groups of properties:

- Message compression
- Temporary destinations
- Connection consumer
- Message format
- Additional

Make any required changes to these groups and then click **Apply** to return to the queue connection factory.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A WebSphere MQ queue connection factory has the following advanced properties.

Compress message headers:

Enables the compression of message headers.

Data type Checkbox
Default Cleared
Range
Cleared
Do not compress message headers.
Selected
Compress message headers.

Compression algorithm for message payloads:

The compression algorithm to use to compress message payloads.

Data type Drop-down list
Default NONE
Range
RLE
ZLIBFAST
ZLIBHIGH
NONE

WebSphere MQ model queue name:

The model queue that is used as a basis for temporary queue definitions.

Data type String
Default SYSTEM.DEFAULT.MODEL.QUEUE

Temporary queue prefix:

The prefix to append to the beginning of the names generated for temporary queues.

Data type String

Retain messages, even if no matching consumer is available:

Determines whether messages for which there is no matching consumer are retained on the input queue or dealt with according with their disposition options.

Data type Checkbox
Default Selected
Range
Cleared
Do not retain messages.
Selected
Retain messages.

Polling interval:

This setting is applicable in the client container only. When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS), this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get messages from the WebSphere MQ queue. The setting determines how long a thread waits for a message to arrive at a WebSphere MQ queue before polling another WebSphere MQ queue in the set from which is being asynchronously consumed.

Data type Integer
Units Milliseconds
Default 5000
Range A value greater than zero.

Rescan interval:

When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS) this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get messages from the WebSphere MQ queue. The setting determines how long a thread retrieves messages from a WebSphere MQ queue before switching to consume messages from another WebSphere MQ queue in the set which is being asynchronously consumed from.

Data type Integer
Units Milliseconds
Default 5000
Range A value greater than zero.

Maximum batch size:

The maximum number of messages to remove from a queue before at least one must be delivered to an asynchronous consumer.

Data type Integer
Default 10
Range A value greater than zero.

Coded character set identifier:

The character set to use when encoding strings in the message.

Data type Integer
Default 819
Range A value greater than zero. The coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging multiplatform and platform-specific books Web pages; for example, at http://www.ibm.com/software/integration/wmq/library, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Append an RFH version 2 header to reply messages:

The action to take when replying to a message without an RFH version 2 header.

Data type Checkbox
Default Selected
Range

Selected
Append an RFH version 2 header to reply messages.

Cleared
Do not append an RFH version 2 header to reply messages.

Fail JMS method calls if the WebSphere MQ queue manager is quiescing:

Selected JMS operations fail when the queue manager is put into a quiescing state. This enables the queue manager to quiesce successfully and shutdown

Data type Checkbox
Default Selected
Range

Cleared
Do not fail JMS operations if the queue manager is quiescing.

Selected
Fail JMS operations if the queue manager is quiescing.

Connection pool settings:

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types, like JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection()

request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

For example: click **Resources** → **JDBC** → **JDBC Providers** → ***JDBC_provider*** → **Data Sources** → ***data_source*** → **Connection pool properties**

The path for JMS queue connection factories is:**Resources** → **JMS** → **Queue connection factories** → ***JMS_queue_connection_factory*** → **[Additional properties] Connection pool properties**.

Connection timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds that a connection request waits when there are no connections available in the free pool and no new connections can be created. This usually occurs because the maximum value of connections in the particular connection pool has been reached.

For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. In most cases you should not retry the getConnection() method; if a longer wait time is required you should increase the Connection Timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.

If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Maximum connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown. For example: If the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

Knowing the number of connection pools that can potentially request connections from the backend (such as a DB2 database or a CICS server) helps you determine a value for the Maximum Connections property.

For multiple standalone application servers that use the same data source configuration, or J2C connection factory configuration, a separate physical connection pool exists for each server. If you clone these same application servers, WebSphere Application Server implements a separate connection pool for each clone.

All of these connection pools correspond to the same data source or connection factory configuration. Therefore all of these connection pools can potentially request connections from the same backend resource, at the same time. The single Maximum Connections value that you set on this console panel applies to every one of these connection pools. Consequently, setting a high Maximum Connections value can result in a load of connection requests that overwhelms your backend resource.

Data type	Integer
Default	10
Range	0 to maximum integer If Max Connections is set to 0, the Connection Timeout value is ignored.

Note: For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum connections:

Specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example, if the Minimum Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not automatically create two additional physical connections to reach the Minimum Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout,

until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting Aged Timeout to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached, the application server will not discard the connection until after the transaction is completed and the connection is closed.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	0

Range 0 to max int

Purge policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**.

Data type String

Defaults

- EntirePool for J2C connection factories and JMS-related connection factories
- EntirePool for WebSphere Version 4.0 data sources
- EntirePool for current version data sources that you create through the administrative console
- EntirePool for current version data sources that you script through wsadmin AdminConfig commands, invoking JDBC templates that are built into WebSphere Application Server (For information on the command **createUsingTemplate**, see the information center article ″Commands for the AdminConfig object.″)
- FailingConnectionOnly for data sources that you script in wsadmin without invoking JDBC templates

:

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

WebSphere MQ topic connection factory collection

Use this panel to select a topic connection factory to view or change its configuration properties. The topic connection factory is configured for the WebSphere MQ messaging provider for publish/subscribe messaging with JMS topics.

This panel shows a list of WebSphere MQ topic connection factories with a summary of their configuration properties.

To view the WebSphere MQ topic connection factory collection, use the administrative console to complete the following steps:

1. In the navigation pane, expand **Resources** → **JMS** → **Topic connection factories** to display existing topic connection factories.
2. If appropriate, in the content pane, change the **Scope** to restrict the set of topic connection factories displayed. If**Scope** is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. Optional: To define a new connection factory, click **New** to start the Create Connection Factory wizard.
4. Optional: To view or change the properties of a connection factory, click its name in the list displayed.

WebSphere MQ messaging provider topic connection factory settings:

Use this panel to view or change the configuration properties of the selected topic connection factory for use with the WebSphere MQ messaging provider. These configuration properties control how connections are created to the associated JMS topic destination.

A WebSphere MQ topic connection factory is used to create JMS connections to topic destinations provided by WebSphere MQ for publish/subscribe messaging.

To view WebSphere MQ topic connection factory settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Topic connection factories** to display existing topic connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the topic connection factories are defined. This restricts the set of topic connection factories displayed.
3. Click the name of the topic connection factory that you want to work with.

Under General Properties there are three groups of properties:
- Administration
- Connection
- Advanced

There are two variations of the **Connection** group:
- CCDT Connection settings group: A WebSphere MQ administrator can create a single Client Channel Definition Table (CCDT) of all the WebSphere MQ channels supported by queue managers in their enterprise. Additional WebSphere MQ configuration information can be associated with the entries in a CCDT. If the selected connection factory was created using a CCDT, only the following information is required to configure a connection:
  - Client channel definition table URL: The URL that specifies the location of the CCDT.
  - Queue manager: The queue manager name that specifies the CCDT entry, or entries, to use.
  - SSL configuration
- Explicitly-defined Connection settings group: If the selected connection factory was not created using a CCDT, the following information must be entered explicitly to configure a connection:
  - Queue manager
  - Transport
  - Hostname
  - Port
  - Server connection channel
  - Use SSL to secure communication with Websphere MQ: If you clear this property, the following properties cannot be used.
    - Centrally managed
    - Specific configuration
    - SSL configuration

Make any required changes to the Administration, Connection and Advanced groups of properties and then click **Apply** before, in the content pane under Additional Properties, you click any of the following links:
- **Advanced properties** to display the advanced properties of your WebSphere MQ topic connection factory
- **Broker properties** to display the broker properties of your WebSphere MQ topic connection factory
- **Custom properties** to display the custom properties of your WebSphere MQ topic connection factory

- **Client transport properties** to display the client transport properties of your WebSphere MQ topic connection factory. This link is only present on the explicitly-specified variation of this panel, where you enter all information required to connect to WebSphere MQ. This link does not appear for the CCDT-based variation.
- **Connection pools** to display the connection pools detail of your WebSphere MQ topic connection factory
- **Session pools** to display the session pools detail of your WebSphere MQ topic connection factory

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the connection factory.

You can specify the following additional properties using WebSphere MQ administrative commands:
- **localAddress**
- **clonedSubs**
- **containerAuthAlias**

For more information about these properties, refer to the “createWMQConnectionFactory command” on page 1966.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

A WebSphere MQ topic connection factory has the following properties.

For more information about setting the SSL properties for WebSphere MQ, see the section SSL properties in the *WebSphere MQ Using Java* book.

Scope:

Specifies the level to which this resource definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource at a different scope, change the scope on the WebSphere MQ topic connection factory collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the topic connection factory is created.

For all topic connection factories using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which this topic connection factory is known for administrative purposes within WebSphere Application Server.

Data type String
Range The name must be unique within the set of JMS topic connection factories defined to the cell.

JNDI name:

The JNDI name that is used to bind the topic connection factory into the JNDI name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of this topic connection factory for administrative purposes within WebSphere Application Server.

Data type String
Default Null

Client channel definition table URL:

A URL that points to a WebSphere MQ CCDT.

Data type String

Queue manager:

If this topic connection factory is based on a CCDT, this property is used to select an entry in the CCDT. Otherwise, it is the name of the queue manager or queue sharing group to connect to. A connection is established to this WebSphere MQ resource to send or receive messages.

Data type String
Range If this topic connection factory is based on a CCDT, the value must be one of the following:

- A valid queue manager name
- A valid queue manager name with the last character an asterisk
- An asterisk
- Blank

If this topic connection factory is not based on a CCDT, the value must be a valid queue manager name.

Transport:

The WebSphere MQ transport type for the connection. This is used to determine the exact mechanisms used to connect to WebSphere MQ.

Data type Drop-down list

Default **bindings then client**

Range

client Use a TCP/IP based network connection to communicate with the WebSphere MQ queue manager

bindings then client
Attempt a bindings mode connection to the queue manager. If this is not possible, revert to the client transport.

bindings
Establish a cross memory connection to a queue manager running on the same node. The following Client Transport Mode properties are disabled:
- Hostname
- Port
- Server connection channel

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to "Configuring the WebSphere MQ messaging provider with native libraries information" on page 1857 and "Sizing the thread pools used by the WebSphere MQ messaging provider" on page 1857.

Hostname:

The hostname, IPv4 or IPv6 address of the WebSphere MQ queue manager to connect to.

Data type String

Port:

The port number on which WebSphere MQ is listening.

Data type Integer

Default 1414

Range The value must be in the range 1 to 65536 (inclusive).

Server connection channel:

The WebSphere MQ server connection channel name used when connecting to WebSphere MQ.

Data type String

Default SYSTEM.DEF.SVRCONN

Range The value must be a server connection channel defined to the WebSphere MQ queue manager being connected to.

Use SSL to secure communications with WebSphere MQ:

This option determines whether the SSL (Secure Sockets Layer) protocol is used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

When using a WebSphere MQ messaging provider topic connection factory in the application server environment, the application server manages SSL configuration. To change SSL configuration parameters, use the administrative console to navigate to the **Security** → **SSL certificate and key management** panel.

When using a WebSphere MQ messaging provider topic connection factory in the client environment, the client takes SSL configuration information from the ssl.client.props file. Use of this file is detailed in the related reference information for this topic.

You can only use one cipher suite in the SSL configuration for a WebSphere MQ messaging provider topic connection factory. If you specify more than one cipher suite, only the first one is used.

Data type

Checkbox. If this checkbox is cleared, the following SSL properties are disabled:

- Centrally managed
- Specific configuration
- SSL configuration

Centrally managed:

When the SSL protocol is used to communicate with WebSphere MQ, select this radio button to specify that the SSL configuration is taken from the centrally managed WebSphere Application Server SSL configuration.

When you select this radio button, the hostname and port attributes from the WebSphere MQ messaging provider topic connection factory are used to select an appropriate SSL configuration. To provide the SSL configuration which will be matched to the topic connection factory, see the Dynamic outbound endpoint SSL configuration settings topic listed under related reference.

Data type

Radio button

Specific configuration:

Select this radio button when you want to specify a particular SSL configuration for use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

Data type

Radio button

SSL configuration:

The specific SSL configuration to use when SSL is to be used to secure network communications with the WebSphere MQ queue manager or queue sharing group.

This property is disabled if the **Centrally managed** radio button is selected and the WebSphere MQ messaging provider resource has been explicitly defined.

This property is always enabled if the WebSphere MQ messaging provider resource is based on a CCDT.

If this WebSphere MQ messaging provider resource is based on a CCDT, this parameter is only used if the relevant entries in the CCDT have been configured to use SSL.

Additionally, if a SSL configuration of `none` is selected, the default centrally managed WebSphere Application Server SSL configuration for the WebSphere MQ messaging provider is used.

Data type Drop-down list

Component-managed authentication alias:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ.

Data type Drop-down list
Default (none)
Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ.

Client identifier:

The client identifier to specify when connecting to WebSphere MQ messaging provider.

Data type String

Allow cloned durable subscriptions:

Determines whether multiple instances of a durable subscription can be accessed concurrently by different servers.

Data type Checkbox
Default Cleared
Range
Selected
Multiple instances of a durable subscription can be accessed concurrently by different servers.
Cleared
Multiple instances of a durable subscription can not be accessed concurrently by different servers.

Provider version:

The WebSphere MQ messaging provider version. This is used to determine whether to connect to a particular version of a queue manager. It is also used to determine the type of functionality required by the client.

Data type String
Range The value entered must be either the empty string or be must be in one of the following formats: n.n.n.n or n.n.n or n.n or n, where n is a numeric value greater than or equal to zero.

Support distributed two phase commit protocol:

Specifies whether the connection factory supports XA coordination of messaging transactions. Enable this option if multiple resources, including this connection factory, are to be used in the same transaction.

If you clear this property, you disable support for distributed two phase commit protocol. The JMS session can still be enlisted in a transaction, but it uses the resource manager local transaction calls (session.commit and session.rollback) instead of XA calls. This can lead to an improvement in performance. However, only a single resource can be enlisted in a transaction in WebSphere Application Server.

Last participant support enables you to enlist one non-XA resource with other XA-capable resources.

Data type Checkbox
Default Selected
Range
Selected
The connection factory supports the use of distributed two phase commit protocols for the coordination of transacted work.
Cleared
The connection factory does not support the use of distributed two phase commit protocols for the coordination of transacted work.
Recommended Do not enable XA when the queue manager specified by the topic connection factory is the only resource in the transaction. Enable XA if transactions involve other resources, including other queues or topics.

Authentication alias for XA recovery:

The authentication alias which specifies the user name and password to use when connecting to WebSphere MQ during XA recovery.

Data type Drop-down list
Default (none)
Range All authentication aliases defined to the cell and the value ″(none)″, which specifies that no credentials are passed to WebSphere MQ during XA recovery.

Mapping-configuration alias:

This field if used only in the absence of a login configuration on the component resource reference.

When the resource authority value is ″container″, the preferred way to define the authentication strategy is by specifying a login configuration and associated properties on the component resource reference.

If the **DefaultPrincipalMapping** login configuration is specified, the associated property will be a JAAS - J2C authentication data entry alias. To configure authentication information for use with the connection factory, under Related items, click **JAAS - J2C authentication data** .

Data type Drop-down list
Default (none)
Range
ClientContainer
WSLogin
WSKRB5Login
DefaultPrincipalMapping
TrustedConnectionMapping
KerberosMapping

WebSphere MQ messaging provider topic connection factory advanced properties:

Use this panel to view or change the advanced properties of the selected topic connection factory for use with the WebSphere MQ messaging provider. These advanced properties control the behavior of connections made to WebSphere MQ messaging provider destinations.

To view WebSphere MQ topic connection factory advanced properties, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Topic connection factories** to display existing topic connection factories.
2. If appropriate, in the content pane, change the **Scope** setting to the level at which the topic connection factories are defined. This restricts the set of topic connection factories displayed.
3. Click the name of the topic connection factory that you want to work with.
4. In the content pane, under Additional properties, click **Advanced properties** to view a list of the advanced properties of the WebSphere MQ topic connection factory.

Under General Properties there are five groups of properties:
- Message compression
- Temporary destinations
- Connection consumer
- Message format
- Additional

Make any required changes to these groups and then click **Apply** to return to the topic connection factory.

A WebSphere MQ topic connection factory has the following advanced properties.

Note:
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book which is available from the WebSphere MQ library page at http://www.ibm.com/software/integration/wmq/library/.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Compress message headers:

Enables the compression of message headers.

Data type Checkbox
Default Cleared
Range
Cleared
Do not compress message headers.
Selected
Compress message headers.

Compression algorithm for message payloads:

The compression algorithm to use to compress message payloads.

Data type Drop-down list
Default NONE
Range
RLE
ZLIBFAST
ZLIBHIGH
NONE

Temporary topic prefix:

The prefix to append to the beginning of the names generated for temporary topics.

Data type	String

Polling interval:

This setting is applicable in the client container only. When using a WebSphere MQ version 6 queue manager (or WebSphere MQ version 5.3 for z/OS), this setting configures the mechanism used to dispatch messages to JMS asynchronous consumers. It is used when the set of WebSphere MQ queues being asynchronously consumed from exceeds the number of threads available internally to synchronously get messages from the WebSphere MQ queue. The setting determines how long a thread waits for a message to arrive at a WebSphere MQ queue before polling another WebSphere MQ queue in the set from which is being asynchronously consumed.

Data type	Integer
Units	Milliseconds
Default	5000
Range	A value greater than zero.

Maximum batch size:

The maximum number of messages to remove from a queue before at least one must be delivered to an asynchronous consumer.

Data type	Integer
Default	10
Range	A value greater than zero.

Coded character set identifier:

The character set to use when encoding strings in the message.

Data type	Integer
Default	819
Range	A value greater than zero. The coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ library page at http://www.ibm.com/software/integration/wmq/library, or the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Fail JMS method calls if the WebSphere MQ messaging provider queue manager is quiescing:

Selected JMS operations fail when the queue manager is put into a quiescing state. This enables the queue manager to quiesce successfully and shutdown.

Data type	Checkbox
Default	Selected

Range

Cleared
Do not fail JMS operations if the queue manager is quiescing.

Selected
Fail JMS operations if the queue manager is quiescing.

Connection pool settings:

Use this page to configure connection pool settings.

This administrative console page is common to a range of resource types, like JDBC data sources and JMS queue connection factories. To view this page, the path depends on the type of resource, but generally you select an instance of the resource provider, then an instance of the resource type, then click **Connection Pool**.

Note: Connection pooling is not supported in an application client. The application client calls the database directly and does not go through a data source. If you want to use the getConnection() request from the application client, configure the JDBC provider in the application client deployment descriptors, using Rational Application Developer or an assembly tool. The connection is established between application client and the database. Application clients do not have a connection pool, but you can configure JDBC provider settings in the client deployment descriptors.

For example: click **Resources** → **JDBC** → **JDBC Providers** → ***JDBC_provider*** → **Data Sources** → ***data_source*** → **Connection pool properties**

The path for JMS queue connection factories is:**Resources** → **JMS** → **Queue connection factories** → ***JMS_queue_connection_factory*** → **[Additional properties] Connection pool properties**.

Connection timeout:

Specifies the interval, in seconds, after which a connection request times out and a ConnectionWaitTimeoutException is thrown.

This value indicates the number of seconds that a connection request waits when there are no connections available in the free pool and no new connections can be created. This usually occurs because the maximum value of connections in the particular connection pool has been reached.

For example, if Connection Timeout is set to 300, and the maximum number of connections are all in use, the pool manager waits for 300 seconds for a physical connection to become available. If a physical connection is not available within this time, the pool manager initiates a `ConnectionWaitTimeout` exception. In most cases you should not retry the getConnection() method; if a longer wait time is required you should increase the Connection Timeout setting value. If a `ConnectionWaitTimeout` exception is caught by the application, review the expected connection pool usage of the application and tune the connection pool and database accordingly.

If the Connection Timeout is set to 0, the pool manager waits as long as necessary until a connection becomes available. This happens when the application completes a transaction and returns a connection to the pool, or when the number of connections falls below the value of Maximum Connections, allowing a new physical connection to be created.

If Maximum Connections is set to 0, which enables an infinite number of physical connections, then the Connection Timeout value is ignored.

Data type Integer

Units	Seconds
Default	180
Range	0 to max int

Maximum connections:

Specifies the maximum number of physical connections that you can create in this pool.

These are the physical connections to the backend resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeoutException is thrown. For example: If the Max Connections value is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in Connection Timeout for a physical connection to become free.

Knowing the number of connection pools that can potentially request connections from the backend (such as a DB2 database or a CICS server) helps you determine a value for the Maximum Connections property.

For multiple standalone application servers that use the same data source configuration, or J2C connection factory configuration, a separate physical connection pool exists for each server. If you clone these same application servers, WebSphere Application Server implements a separate connection pool for each clone.

All of these connection pools correspond to the same data source or connection factory configuration. Therefore all of these connection pools can potentially request connections from the same backend resource, at the same time. The single Maximum Connections value that you set on this console panel applies to every one of these connection pools. Consequently, setting a high Maximum Connections value can result in a load of connection requests that overwhelms your backend resource.

Data type	Integer
Default	10
Range	0 to maximum integer If Max Connections is set to 0, the Connection Timeout value is ignored.

Note: For better performance, set the value for the connection pool lower than the value for the Max Connections option in the Web container. Lower settings, such as 10-30 connections, perform better than higher settings, such as 100.

You can use the Tivoli Performance Viewer to find the optimal number of connections in a pool. If the number of concurrent waiters is greater than 0, but the CPU load is not close to 100%, consider increasing the connection pool size. If the Percent Used value is consistently low under normal workload, consider decreasing the number of connections in the pool.

Minimum connections:

Specifies the minimum number of physical connections to maintain.

If the size of the connection pool is at or below the minimum connection pool size, the Unused Timeout thread does not discard physical connections. However, the pool does not create connections solely to ensure that the minimum connection pool size is maintained. Also, if you set a value for Aged Timeout, connections with an expired age are discarded, regardless of the minimum pool size setting.

For example, if the Minimum Connections value is set to 3, and one physical connection is created, the Unused Timeout thread does not discard that connection. By the same token, the thread does not

automatically create two additional physical connections to reach the Minimum Connections setting.

Data type	Integer
Default	1
Range	0 to max int

Reap time:

Specifies the interval, in seconds, between runs of the pool maintenance thread.

For example, if Reap Time is set to 60, the pool maintenance thread runs every 60 seconds. The Reap Time interval affects the accuracy of the Unused Timeout and Aged Timeout settings. The smaller the interval, the greater the accuracy. If the pool maintenance thread is enabled, set the Reap Time value less than the values of Unused Timeout and Aged Timeout. When the pool maintenance thread runs, it discards any connections remaining unused for longer than the time value specified in Unused Timeout, until it reaches the number of connections specified in Minimum Connections. The pool maintenance thread also discards any connections that remain active longer than the time value specified in Aged Timeout.

The Reap Time interval also affects performance. Smaller intervals mean that the pool maintenance thread runs more often and degrades performance.

To disable the pool maintenance thread set Reap Time to 0, or set both Unused Timeout and Aged Timeout to 0. The recommended way to disable the pool maintenance thread is to set Reap Time to 0, in which case Unused Timeout and Aged Timeout are ignored. However, if Unused Timeout and Aged Timeout are set to 0, the pool maintenance thread runs, but only physical connections which timeout due to non-zero timeout values are discarded.

Data type	Integer
Units	Seconds
Default	180
Range	0 to max int

Unused timeout:

Specifies the interval in seconds after which an unused or idle connection is discarded.

Set the Unused Timeout value higher than the Reap Timeout value for optimal performance. Unused physical connections are only discarded if the current number of connections exceeds the Minimum Connections setting. For example, if the unused timeout value is set to 120, and the pool maintenance thread is enabled (Reap Time is not 0), any physical connection that remains unused for two minutes is discarded.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	1800
Range	0 to max int

Aged timeout:

Specifies the interval in seconds before a physical connection is discarded.

Setting Aged Timeout to 0 supports active physical connections remaining in the pool indefinitely. Set the Aged Timeout value higher than the Reap Timeout value for optimal performance.

For example, if the Aged Timeout value is set to 1200, and the Reap Time value is not 0, any physical connection that remains in existence for 1200 seconds (20 minutes) is discarded from the pool. The only exception is if the connection is involved in a transaction when the aged timeout is reached, the application server will not discard the connection until after the transaction is completed and the connection is closed.

The accuracy and performance of this timeout are affected by the Reap Time value. See “Reap time” on page 1401 for more information.

Data type	Integer
Units	Seconds
Default	0
Range	0 to max int

Purge policy:

Specifies how to purge connections when a *stale connection* or *fatal connection error* is detected.

Valid values are **EntirePool** and **FailingConnectionOnly**.

Data type	String
Defaults	• EntirePool for J2C connection factories and JMS-related connection factories • EntirePool for WebSphere Version 4.0 data sources • EntirePool for current version data sources that you create through the administrative console • EntirePool for current version data sources that you script through wsadmin AdminConfig commands, invoking JDBC templates that are built into WebSphere Application Server (For information on the command **createUsingTemplate**, see the information center article ″Commands for the AdminConfig object.″) • FailingConnectionOnly for data sources that you script in wsadmin without invoking JDBC templates :

Range

EntirePool
All connections in the pool are marked stale. Any connection not in use is immediately closed. A connection in use is closed and issues a stale connection Exception during the next operation on that connection. Subsequent getConnection() requests from the application result in new connections to the database opening. When using this purge policy, there is a slight possibility that some connections in the pool are closed unnecessarily when they are not stale. However, this is a rare occurrence. In most cases, a purge policy of EntirePool is the best choice.

FailingConnectionOnly
Only the connection that caused the stale connection exception is closed. Although this setting eliminates the possibility that valid connections are closed unnecessarily, it makes recovery from an application perspective more complicated. Because only the currently failing connection is closed, there is a good possibility that the next getConnection() request from the application can return a connection from the pool that is also stale, resulting in more stale connection exceptions.

The connection pretest function attempts to insulate an application from pooled connections that are not valid. When a backend resource, such as a database, goes down, pooled connections that are not valid might exist in the free pool. This is especially true when the purge policy is failingConnectionOnly; in this case, the failing connection is removed from the pool. Depending on the failure, the remaining connections in the pool might not be valid.

WebSphere MQ queue destination collection

The JMS queue destinations configured in the WebSphere MQ messaging provider for point-to-point messaging with JMS queues. Use this panel to create or delete queue destinations, or to select a queue destination to view or change its configuration properties.

This panel shows a list of JMS queue destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If **Scope** is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, click the **WebSphere MQ messaging provider** that you want to support the JMS destination.
4. In the content pane, under Additional Properties, click **Queues**. This displays a list of any existing JMS queue destinations.

To create a new queue destination, click **New**.

To view or change the properties of a queue destination, select its name in the list displayed.

To act on one or more of the queue destinations listed, click the check box next to the name of the queue, then use the buttons provided.

WebSphere MQ messaging provider queue settings:

Use this panel to view or change the configuration properties of the selected queue destination for point-to-point messaging with WebSphere MQ as a messaging provider.

A WebSphere MQ queue destination is used to configure the properties of a queue for the WebSphere MQ messaging provider. Connections to the queue are created using an associated WebSphere MQ queue connection factory or connection factory.

To view WebSphere MQ queue settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Queues** to display existing queue destinations.
2. Click the name of the queue destination that you want to work with.
3. Optional: To create a new queue destination, click **New**.
4. Optional: To view or change the queue destination settings, click its name in the list displayed.

Under General Properties there are two groups of properties:
- Administration
- WebSphere MQ Queue

Make any required changes to the Administration and WebSphere MQ Queue groups of properties and then click **Apply** before, in the content pane under Additional Properties, you click one of the following links:
- **Advanced properties** to display or change the advanced properties of your WebSphere MQ queue destination
- **WebSphere MQ queue connection properties** to display or change the connection properties of your WebSphere MQ queue destination
- **Custom properties** to display or change the custom properties of your WebSphere MQ queue destination

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the queue destination.

Note: When specifying WebSphere MQ properties, the following restrictions apply:
- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

A queue destination for use with the WebSphere MQ messaging provider has the following properties.

Scope:

The scope assigned to the queue when it is created. The scope specifies the level to which this queue definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource at a different scope, change the scope on WebSphere MQ queue destination collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the queue is created.

For all queues created using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which the queue is known for administrative purposes within WebSphere Application Server.

Data type String

JNDI name:

The name that is used to bind the queue into the JNDI name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of the queue, for administrative purposes within WebSphere Application Server.

Data type String
Default Null

Queue name:

The WebSphere MQ name for the queue that holds the messages for the JMS destination.

Data type String

Queue manager or queue sharing group name:

The name of the WebSphere MQ queue manager or queue sharing group where the queue resides.

Data type String

WebSphere MQ queue connection properties:

Use this panel to specify how to connect to the queue manager that hosts the queue.

The system uses these connection properties to retrieve, display and update the queue configuration details that are shown on the **WebSphere MQ queue settings** panel.

To set of change the queue connection properties, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Queues** to display existing queue destinations.
2. Click the name of the queue destination that you want to work with.
3. To view or change the queue destination settings, click its name in the list displayed.

Under General Properties there are two groups of properties:
- Administration
- WebSphere MQ Queue

Make any required changes to the Administration and WebSphere MQ Queue groups of properties and then click **Apply** before, in the content pane under Additional Properties, you click the **WebSphere MQ queue connection properties** link to display or change the connection properties of your WebSphere MQ queue destination.

Make any required changes to the General properties and then click **Apply** before, in the content pane under Additional Properties, you click **WebSphere MQ configuration** to return to the **WebSphere MQ queue settings** panel.

A queue destination for use with the WebSphere MQ messaging provider has the following WebSphere MQ queue connection properties.

Note:
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Queue Manager Host:

The name of host for the queue manager on which the queue destination is created.

Data type	String

Queue Manager Port:

The number of the port used by the queue manager on which this queue is defined.

Data type	Integer
Range	A valid TCP/IP port number. This port must be configured on the WebSphere MQ queue manager.

Server Connection Channel Name:

The name of the channel used for connection to the WebSphere MQ queue manager.

Data type	String
Range	1 through 20 ASCII characters

User ID:

The user ID used, with the **Password** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.*

Data type String

Password:

The password, used with the **User ID** property, for authentication when connecting to the queue manager to define the queue destination.

*If you specify a value for the **User ID** property, you must also specify a value for the **Password** property.*

Data type String

WebSphere MQ topic destination collection

The JMS topic destinations configured in the WebSphere MQ messaging provider for point-to-point messaging with JMS topics. Use this panel to create or delete topic destinations, or to select a topic destination to view or change its configuration properties.

This panel shows a list of JMS topic destinations with a summary of their configuration properties.

To view this administrative console page, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **JMS providers**.
2. If appropriate, in the content pane, change the scope of the WebSphere MQ messaging provider. If **Scope** is set to node or server scope for a Version 5 node, the administrative console presents the subset of resources and properties that are applicable to WebSphere Application Server Version 5.
3. In the content pane, click the name of the WebSphere MQ messaging provider that you want to support the JMS destination.
4. In the content pane, under Additional Properties, click **Topics**. This displays a list of any existing JMS topic destinations.

To create a new topic destination, click **New**.

To view or change the properties of a topic destination, select its name in the list displayed.

To act on one or more of the topic destinations listed, select the check boxes next to the names of the objects that you want to act on, then use the buttons provided.

WebSphere MQ messaging provider topic settings:

Use this panel to view or change the configuration properties of the selected JMS topic destination for publish/subscribe messaging with WebSphere MQ as a messaging provider.

A WebSphere MQ topic destination is used to configure the properties of a topic for the WebSphere MQ messaging provider. Connections to the topic are created using an associated WebSphere MQ topic connection factory or connection factory.

To view WebSphere MQ topic settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS** → **Topics** to display existing topic destinations.

2. Click the name of the topic destination that you want to work with.
3. Optional: To create a new topic destination, click **New**.
4. Optional: To view or change the topic destination settings, click its name in the list displayed.

Under General Properties there are two groups of properties:

- Administration
- WebSphere MQ topic

Make any required changes to the Administration and WebSphere MQ topic groups of properties and then click **Apply** before, in the content pane under Additional Properties, you click either of the following links:

- **Advanced properties** to display or change the advanced properties of your WebSphere MQ topic
- **Custom properties** to display or change the custom properties of your WebSphere MQ topic

Under Related items, click **JAAS - J2C authentication data** to configure authentication information for use with the topic.

You can specify the following additional properties using WebSphere MQ administrative commands:

- **wildcardFormat**
- **brokerVersion**

For more information about these properties, refer to the “createWMQTopic command” on page 1981.

A topic for use with the WebSphere MQ messaging provider has the following properties.

Note:

- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ JMS resources, see the WebSphere MQ *Using Java* book and the *WebSphere MQ System Administration* book, SC33-1873, which are available from the WebSphere MQ messaging platform-specific books Web page at the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.
- In WebSphere MQ, names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.

Scope:

The scope assigned to the topic when it is created. The scope specifies the level to which this topic definition is visible to applications.

Resources such as messaging providers, namespace bindings, or shared libraries can be defined at multiple scopes, with resources defined at more specific scopes overriding duplicates which are defined at more general scopes.

The scope displayed is for information only, and cannot be changed on this panel. If you want to browse or change this resource at a different scope, change the scope on the WebSphere MQ topic destination collection panel, then click **Apply**, before clicking the link for the type of resource.

Data type String

Provider:

The JMS provider assigned when the topic is created.

For all topics created using this panel, the provider is the WebSphere MQ messaging provider.

The provider is displayed for information only.

Data type String

Name:

The name by which the topic is known for administrative purposes within IBM WebSphere Application Server.

Data type String

JNDI name:

The name that is used to bind the topic into the JNDI name space.

As a convention, use the fully qualified JNDI name; for example, in the form jms/*Name*, where *Name* is the logical name of the resource.

Data type String

Description:

A description of the topic for administrative purposes within IBM WebSphere Application Server.

Data type String

Topic name:

The WebSphere MQ name for the topic.

Data type String

Broker durable subscriber queue:

The queue to use for durable subscribers.

Data type Drop-down list
Default As connection
Range
As connection
The connection factory values are used.
Override connection
Enter the name of a queue in the associated text box.

Broker durable subscription connection consumer queue:

The queue to use for durable subscribers that use a connection consumer.

Data type Drop-down list
Default As connection

Range

As connection
The connection factory values are used.
Override connection
Enter the name of a queue in the associated text box.

Broker publication queue:

The queue to use for publishing messages.

Data type Drop-down list
Default As connection
Range

As connection
The connection factory values are used.
Override connection
Enter the name of a queue in the associated text box.

Broker publication queue manager:

The queue manager that hosts the topic.

Data type Drop-down list
Default As connection
Range

As connection
The connection factory values are used.
Override connection
Enter the name of a queue manager in the associated text box.

WebSphere MQ messaging provider queue and topic advanced properties settings:

Use this panel to view or change the advanced properties for the selected queue or topic destination for messaging with the WebSphere MQ messaging provider.

To view WebSphere MQ queue or topic advanced properties settings, use the administrative console to complete the following steps:
1. In the navigation pane, expand **Resources** → **JMS**.
2. Click **Queues** or **Topics** to display existing queue or topic destinations.
3. If appropriate, in the content pane, change the **Scope** setting to the level at which the queue or topic destinations are defined. This restricts the set of queue or topic destinations displayed.
4. Click the name of the queue or topic destination that you want to work with.
5. In the content pane, under Additional properties, click **Advanced properties** to display a list of the advanced properties of the WebSphere MQ queue or topic destination.

Under General Properties there are three groups of properties:
- Delivery
- Message format
- Optimizations

Make any required changes to these groups and then click **Apply** to return to the queue or topic.

Note: When specifying WebSphere MQ properties, the following restrictions apply:

- Names can have a maximum of 48 characters, with the exception of channels which have a maximum of 20 characters.
- The property values that you specify must match the values that you specified when configuring WebSphere MQ for JMS resources. For more information about configuring WebSphere MQ for JMS resources, see the WebSphere MQ *Using Java* book at http://www.ibm.com/software/integration/wmq/library/.

A queue or topic for use with the WebSphere MQ messaging provider has the following advanced properties.

Persistence:

The level of persistence used to store messages sent to this destination.

Data type Drop-down list
Default As set by application
Range
As set by application
Messages on the destination have their persistence defined by the application that put them onto the queue.
As for WebSphere MQ queue definition
Messages on the destination have their persistence defined by the WebSphere MQ queue definition properties.
WebSphere MQ Persistent
Messages on the destination are persistent.
WebSphere MQ Non-persistent
Messages on the destination are not persistent.
WebSphere MQ High
Permit persistent messages to be sent as non-persistent messages when using an underlying WebSphere MQ queue with a NPMCLASS of 'HIGH'.

Priority:

The priority assigned to messages sent to this destination.

Data type Drop-down list
Default As set by application
Range
As set by application
The priority of messages on this destination is defined by the application that put them onto the destination.
As for WebSphere MQ queue definition
Messages on the destination have their persistence defined by the WebSphere MQ destination definition properties.
Specified
The priority of messages on this destination is defined by the **Specified priority** property. If you select this option, you must define a priority on the **Specified priority** property.

Specified priority:

If the **Priority** property was set to `Specified`, select the priority assigned to messages sent to this queue type destination.

Data type	Drop-down list
Units	Message priority level
Default	As set by application
Range	0 (lowest priority) through 9 (highest priority)

Expiry:

Whether the expiry timeout for this destination is defined by the application or the **Specified expiry** property, or messages on the destination never expire.

Data type	Drop-down list
Default	As set by application
Range	**As set by application** The expiry timeout for messages on this destination is defined by the application that put them onto the destination. **Specified** The expiry timeout for messages on this destination is defined by the **Specified expiry** property. If you select this option, you must define a timeout on the **Specified expiry** property. **Unlimited** Messages on this destination have no expiry timeout, so those messages never expire.

Specified expiry:

If the **Expiry** property is set to `Specified`, enter the number of milliseconds after which messages expire and are removed from this destination.

Data type	Integer
Units	Milliseconds
Default	0
Range	Greater than or equal to 0 • 0 indicates that messages never timeout • Other values are an integer number of milliseconds

Coded character set identifier:

The character set to use when encoding strings in the message.

Data type	Integer
Default	1208
Range	1 through 65535. The coded character set identifier (CCSID) must be one of the CCSIDs supported by WebSphere MQ.

For more information about supported CCSIDs, and about converting between message data from one coded character set to another, see the *WebSphere MQ System Administration* and the *WebSphere MQ Application Programming Reference* books. These are available from the WebSphere MQ messaging

multiplatform and platform-specific books Web pages; for example, at http://www.ibm.com/software/integration/wmq/library, the IBM Publications Center, or from the WebSphere MQ collection kit, SK2T-0730.

Use native encoding:

Whether or not the destination should use native encoding to provided appropriate encoding values for the Java platform.

Data type Checkbox
Default Selected
Range
Selected
Native encoding is used.
Cleared
Native encoding is not used, so specify the properties for **Integer encoding**, **Decimal encoding**, and **Floating point encoding**.

Integer encoding:

If the **Use native encoding** checkbox is cleared, select the type of integer encoding to be used.

Data type Drop-down list
Default Normal
Range
Normal
Normal integer encoding is used.
Reversed
Reversed integer encoding is used.

Decimal encoding:

If the **Use native encoding** is cleared, select the type of decimal encoding to be used.

Data type Drop-down list
Units Not applicable
Default Normal
Range
Normal
Normal decimal encoding is used.
Reversed
Reversed decimal encoding is used.

Floating point encoding:

If the **Use native encoding** is cleared, select the type of floating point encoding to be used.

Data type Drop-down list
Default IEEENORMAL
Range
IEEENORMAL
IEEE normal floating point encoding is used.
IEEEREVERSED
IEEE reversed floating point encoding is used.
z/OS z/OS floating point encoding is used.

Append RFH version 2 headers to messages sent to this destination:

Whether RFH version 2 headers should be appended to messages sent to this destination.

Data type Checkbox
Default Selected
Range
Cleared
Do not append RFH version 2 headers to messages sent to this destination.
Selected
Append RFH version 2 headers to messages sent to this destination.

Asynchronously send messages to the queue manager:

Whether the queue manager acknowledges receipt of messages sent to it. Asynchronously sending messages to the queue manager is faster, but messages can be lost if the messaging infrastructure fails.

Data type Drop-down list
Default
The default value depends on whether you are working with a queue or topic destination.
As for queue definition
This is the default value if you are working with a queue destination.
As for topic definition
This is the default value if you are working with a topic destination.
Range
As for queue definition
Messages are acknowledged according to the WebSphere MQ queue definition properties.
As for topic definition
Messages are acknowledged according to the WebSphere MQ topic definition properties.
Yes The queue manager acknowledges receipt of messages sent to it.
No The queue manager does not acknowledge receipt of messages sent to it.

Read ahead and cache non-persistent messages for consumers:

Whether messages for non-persistent consumers are sent to the client speculatively. This results in faster message delivery but messages can be lost in the event of a failure in the messaging infrastructure.

Data type Drop-down list
Default
The default value depends on whether you are working with a queue or topic destination.
As for queue definition
This is the default value if you are working with a queue destination.
As for topic definition
This is the default value if you are working with a topic destination.
Range
As for queue definition
Messages are sent to the client according to the WebSphere MQ queue definition properties.
As for topic definition
Messages are sent to the client according to the WebSphere MQ topic definition properties.
Yes Messages are sent to the client speculatively.
No Messages are not sent to the client speculatively.

Read ahead consumer close method:

If **Read ahead and cache non-persistent messages for consumers** is set to `Yes` or `As for queue definition` this property is enabled. Determines what happens to messages in the internal read ahead buffer when the message consumer is closed.

Data type	Drop-down list
Default	Close method waits for all cached messages to be delivered
Range	**Wait for all cached messages to be delivered** All messages in the internal read ahead buffer are delivered to the application's message listener before returning. **Wait for the current message to be delivered** Only the current message listener invocation completes before returning, potentially leaving messages in the internal read ahead buffer, which are then discarded.

WMQAdminCommands command group for the AdminTask object

You can use the WebSphere MQ administrative commands to manage JMS resources for the WebSphere MQ messaging provider.

You can configure JMS resources for the WebSphere MQ messaging provider through the WebSphere MQ administrative commands, or you can configure JMS resources for the WebSphere MQ messaging provider through the administrative console.

To run these commands, use the AdminTask object of the wsadmin scripting client. Each command acts on multiple objects in one operation. The commands are provided to allow you to make the most commonly-required types of update in a consistent manner, where modifying the underlying objects directly would be error-prone.

The wsadmin scripting client is run from Qshell. For more information, see the topic "Configure Qshell to run WebSphere Application Server scripts".

These commands are valid only when they are used with WebSphere Application Server Version 7 and later application servers. Do not use them with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using these commands, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

The following commands are available for the WMQAdminCommands group of the AdminTask object:

- createWMQActivationSpec command

- deleteWMQActivationSpec command
- listWMQActivationSpec command
- modifyWMQActivationSpec command
- showWMQActivationSpec command
- createWMQConnectionFactory command
- deleteWMQConnectionFactory command
- listWMQConnectionFactory command
- modifyWMQConnectionFactory command
- showWMQConnectionFactory command
- createWMQTopic command
- deleteWMQTopic command
- listWMQTopic command
- modifyWMQTopic command
- showWMQTopic command
- manageWMQ command
- migrateWMQMLP command
- createWMQQueue command
- deleteWMQQueue command
- listWMQQueue command
- modifyWMQQueue command
- showWMQQueue command

createWMQActivationSpec command

Use the createWMQActivationSpec command to create an activation specification for the WebSphere MQ messaging provider at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the createWMQActivationSpec command to create a WebSphere MQ messaging provider activation specification at a specific scope.

You cannot create a WebSphere MQ messaging provider activation specification under either of the following conditions:

- A WebSphere MQ messaging provider activation specification already exists with the same name, at the same scope.
- The JNDI name clashes with another entry in WebSphere Application Server JNDI.

You create a CCDT based activation specification by specifying any of the following parameters:

- **-ccdtUrl**
- **-ccdtQmgrName**

If you do not specify any of the following parameters, you create a generic activation specification :

- **-ccdtUrl**
- **-ccdtQmgrName**

Target object

The scope of the WebSphere MQ messaging provider at which the WebSphere MQ messaging provider activation specification is to be created.

Required parameters

-name
The administrative name assigned to this WebSphere MQ messaging provider activation specification.

-jndiName
The name and location used to bind this object into WebSphere Application Server JNDI.

-destinationJndiName
The JNDI name of a WebSphere MQ messaging provider queue or topic type destination. When an MDB is deployed with this activation specification, messages for the MDB are consumed from this destination.

-destinationType
The type of the destination specified using the **-destinationJndiName** parameter.

Enter one of the following values:

- `javax.jms.Queue`
- `javax.jms.Topic`

There is no default value.

Optional parameters

-description
An administrative description assigned to the activation specification.

-ccdtUrl
A URL to a client channel definition table to use, for this activation specification, when contacting WebSphere MQ.

Use this parameter to create ccdtURL activation specifications

Do not specify this parameter in conjunction with the following parameters: **-qmgrName**, **-qmgrType**, **-qmgrHostname**, **-qmgrPortNumber**, **-qmgrSvrconnChannel**, or **-localAddress**.

-ccdtQmgrName
A queue manager name, used to select one or more entries from a client channel definition table.

You must specify this parameter if the **-transportType** has been specified as `client` or `bindingsThenClient`.

Do not specify this parameter in conjunction with the following parameters: **-qmgrName**, **-qmgrType**, **-qmgrHostname**, **-qmgrPortNumber**, **-qmgrSvrconnChannel**, or **-localAddress**.

-qmgrName
The name of the queue manager to use, for this activation specification, when contacting WebSphere MQ.

Use this parameter to create generic activation specifications.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-wmqTransportType
This parameter determines the way in which a connection is established to WebSphere MQ for this activation specification.

Use this parameter to create generic activation specifications.

Enter one of the following case-sensitive values:

- `BINDINGS`
- `BINDINGS_THEN_CLIENT`
- `CLIENT`

`BINDINGS_THEN_CLIENT` is the default value.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to “Configuring the WebSphere MQ messaging provider with native libraries information” on page 1857and “Sizing the thread pools used by the WebSphere MQ messaging provider” on page 1857.

-qmgrHostname
The host name to use, for this activation specification, when attempting a client mode connection to WebSphere MQ. It must be a valid TCP/IP hostname or IPv4 or IPv6 address.

The default value is the local host.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-qmgrPortNumber
The port number to use, for this activation specification, when attempting a client mode connection to WebSphere MQ.

Enter an integer value in the range 1 – 65536 (inclusive).

The default value is 1414.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-authAlias
The authentication alias used to obtain the credentials specified when this activation specification needs to establish a connection to WebSphere MQ.

-clientId
The client identifier used for connections started using this activation specification.

-providerVersion
This parameter determines the minimum version, and capabilities of the queue manager.

Enter values in one of the following formats:

- *n*
- *n.n*
- *n.n.n*
- *n.n.n.n*

where *n* is an integer greater than or equal to zero.

-sslCrl
This parameter specifies a list of LDAP servers that are used to provide certificate revocation information if this activation specification establishes an SSL based connection to WebSphere MQ.

-sslResetCount
This parameter is used when the activation specification establishes an SSL connection to the queue manager. This parameter determines how many bytes to transfer before resetting the symmetric encryption key that is used for the SSL session.

Enter a value in the range 0 through 999,999,999.

The default value is 0.

-sslPeerName
This parameter is used when the activation specification establishes an SSL connection to the queue manager. The value is compared with the distinguished name present in the peer's certificate.

-rcvExit
A comma-separated list of receive exit class names.

-rcvExitInitData
Initialization data to pass to the receive exit.

Do not specify this parameter unless you specify the **-rcvExit** parameter.

-sendExit
A comma-separated list of send exit class names.

-sendExitInitData
Initialization data to pass to the send exit.

Do not specify this parameter unless you specify the **-sendExit** parameter.

-secExit
A security exit class name.

-secExitInitData
Initialization data to pass to the security exit.

Do not specify this parameter unless you specify the **-secExit** parameter.

-compressHeaders
This parameter determines if message headers are compressed.

Enter one of the following values:

- `NONE`
- `SYSTEM`

The default value is `NONE`.

-compressPayload
This parameter determines if message payloads are compressed.

Enter one of the following values:

- NONE
- RLE
- ZLIBFAST
- ZLIBHIGH

The default value is NONE.

-msgRetention
This parameter determines if the connection consumer keeps unwanted messages on the input queue.

Enter one of the following values:

- YES
- NO

where YES specifies that the connection consumer keeps unwanted messages on the input queue.

and NO specifies that the messages are disposed of according to their disposition options.

The default value is YES.

-rescanInterval
When a message consumer in the point-to-point domain uses a message selector to select which messages it wants to receive, the JMS client searches the WebSphere MQ queue for suitable messages in the sequence determined by the MsgDeliverySequence attribute of the queue. When the client finds a suitable message and delivers it to the consumer, the client resumes the search for the next suitable message from its current position in the queue. The client continues to search the queue in this way until it reaches the end of the queue, or until the interval of time in milliseconds, as determined by the value of this **-rescanInterval** parameter has expired. In each case, the client returns to the beginning of the queue to continue its search, and a new time interval commences.

This parameter must be a positive integer value.

The default value is 5000.

-ccsid
The coded character set identifier to be used on connections.

The value of this parameter must be a positive integer.

The default value is 819.

-failIfQuiescing
This parameter determines the behavior of certain calls to the queue manager when the queue manager is put into quiescing state.

The value of this parameter must be True or False.

True specifies that calls to certain methods fail if the queue manager is in a quiescing state. If an application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop.

False specifies that no methods fail if the queue manager is in a quiescing state. If you specify this value, an application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager, and therefore prevent the queue manager from stopping.

The default value is True.

-brokerCtrlQueue
The name of the broker control queue to use if this activation specification is to subscribe to a topic.

The default value is SYSTEM.BROKER.CONTROL.QUEUE.

-brokerSubQueue
The name of the queue to use for obtaining subscription messages if this activation specification is to subscribe to a topic.

The default value is `SYSTEM.JMS.ND.SUBSCRIBER.QUEUE`.

-brokerCCSubQueue
The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer.

The default value is `SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE`.

-brokerVersion
The value of this parameter determines the level of functionality required for publish/subscribe operations.

Valid values are `1` and `2`.

The default value is `1`.

-msgSelection
This parameter determines where message selection occurs.

Valid values are `CLIENT` and `BROKER`.

The default value is `CLIENT`.

-subStore
This parameter determines where WebSphere MQ messaging provider stores persistent data relating to active subscriptions.

Valid values are `MIGRATE`, `QUEUE` and `BROKER`.

The default value is `MIGRATE`.

-stateRefreshHint
The interval, in milliseconds, between refreshes of the long running transaction that detects when a subscriber loses its connection to the queue manager. This parameter is relevant only if **-subStore** parameter has the value `QUEUE`.

The value of this parameter must be a positive integer.

The default value is `60,000`.

-cleanupLevel
The cleanup level for BROKER or MIGRATE subscription stores.

Valid values are `SAFE`, `ASPROP`, `NONE` and `STRONG`.

The default value is `SAFE`.

-cleanupInterval
The interval between background executions of the publish/subscribe cleanup utility.

The value of this parameter must be a positive integer.

The default value is `3,600,000`.

-wildcardFormat
This parameter determines which sets of characters are interpreted as topic wildcards.

Valid values are `Topic` or `Char`.

`Char` is the default value.

-sparseSubs
This parameter controls the message retrieval policy of a TopicSubscriber object.

The value of this parameter must be `True` or `False`

The default value is `False`.

-brokerQmgr
The name of the queue manager on which the broker is running.

-clonedSubs
This parameter determines whether two or more instances of the same durable topic subscriber can run simultaneously.

The value of this parameter must be `ENABLED` or `DISABLED`

The default value is `ENABLED`.

-qmgrSvrconChannel
The SVRCONN channel to use when connecting to WebSphere MQ.

Use this parameter to create **explicitly defined** activation specifications.

The default value is `SYSTEM.DEF.SVRCONN`.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-brokerCCDurSubQueue
The name of the queue from which a connection consumer receives durable subscription messages.

The default value is `SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE`.

-maxPoolSize
The maximum number of server sessions in the server session pool used by the connection consumer.

The value of this parameter must be a positive integer.

The default value is `10`.

-messageSelector
A message selector expression specifying which messages are to be delivered.

The value of this parameter must be either the empty string or a valid SQL 92 statement.

-poolTimeout
The period of time, in milliseconds, that an unused server session is held open in the server session pool before being closed due to inactivity.

The value of this parameter must be a positive integer.

The default value is `300,000`.

-startTimeout
The period of time, in milliseconds, within which delivery of a message to an MDB must start after the work to deliver the message has been scheduled. If this period of time elapses, the message is rolled back onto the queue.

The value of this parameter must be a positive integer.

The default value is `10,000`.

-subscriptionDurability
This parameter determines whether a durable or nondurable subscription is used to deliver messages to an MDB that is subscribing to the topic.

The value of this parameter must be `Durable` or `NonDurable`

The default value is `NonDurable`.

-subscriptionName
The name of the durable subscription.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider activation specification implementation. Typically, custom properties are used to set attributes of the activation specification which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

-localAddress
This parameter specifies either or both of the following:

- the local network interface
- the local port, or range of local ports

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-sslType
This parameter determines the configuration, if any, to use when applying SSL encryption to the network connection to the queue manager.

The value of this parameter must be `CENTRAL`, `SPECIFIC` or `NONE`

The **-sslConfiguration** parameter is not valid unless this parameter is set to `SPECIFIC`.

The default value is `NONE`.

-sslConfiguration
The name of the SSL configuration to use when using SSL to secure network connections to the queue manager.

Do not specify this parameter unless the parameter **-sslType** is assigned the value `SPECIFIC`.

The value of this parameter must correspond to an SSL configuration.

There is no default value.

-stopEndpointIfDeliveryFails
This parameter indicates whether the endpoint should be stopped if message delivery fails the number of times specified by the **failureDeliveryCount** property.

The value of this parameter must be `true` or `false`.

The default value is `true`.

-failureDeliveryCount
This parameter specifies the number of sequential delivery failures that are allowed before the endpoint is suspended. This value is only used if **stopEndpointIfDeliveryFails** is `true`.

The value of this parameter must be a non-negative integer.

The default value is `0`, which means that the endpoint is stopped the first time it fails.

Minimal activation specification definition

The following example creates an activation specification, specifying the minimum number of parameters. Due to the default values assumed for the unspecified parameters, MDBs deployed using this activation specification are co-located with a generic queue manager installed on the same node.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQActivationSpec("9994GKCNode01(cells/9994GKCNode01Cell/
```

```
nodes/9994GKCNode01|node.xml#Node_1)", ["-name spec1 –jndiName jms/as/spec1
 –destinationJndiName jms/queues/q1 –destinationType javax.jms.Queue"])
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQActivationSpec
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name spec1 –jndiName jms/as/spec1 –destinationJndiName jms/queues/q1
–destinationType javax.jms.Queue}
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

Explicit activation specification definition

The following example creates an activation specification for which the user must specify and maintain all the parameters used for establishing a connection to WebSphere MQ.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQActivationSpec("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)", ["-name spec2 –jndiName 'jms/as/spec2'
 –destinationJndiName 'jms/topics/t2' –destinationType javax.jms.Topic
 –description 'Must remember to keep each of these activation specifications in
 sync with the WebSphere MQ queue manager to which they refer' –qmgrName QM1
 –qmgrHostname 192.168.0.22 –qmgrPort 1415 –qmgrSvrconnChannel QM1.SVRCONN"])
spec2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234987)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQActivationSpec
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name spec2 –jndiName "jms/as/spec2" –destinationJndiName "jms/topics/t2"
–destinationType javax.jms.Topic –description "Must remember to keep each
of these activation specifications in sync with the WebSphere MQ queue manager
to which they refer" –qmgrName QM1 –qmgrHostname 192.168.0.22 –qmgrPort 1415
–qmgrSvrconnChannel QM1.SVRCONN}
spec2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234987)
```

Activation specification definition specifying a CCDT

The following example creates an activation specification that uses a CCDT to locate the queue manager to connect to.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode019994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)")

wsadmin>AdminTask.createWMQActivationSpec("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)", ["-name spec3 –jndiName 'jms/as/spec3'
 –destinationJndiName 'jms/queue/q3' –destinationType javax.jms.Queue
 –ccdtUrl 'http://gorillaaction:9080/ccdt/amqclchl.tab' –ccdtQmgrName QM3"])
spec3(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234988)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQActivationSpec
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name spec3 –jndiName "jms/as/spec3" –destinationJndiName "jms/queue/q3"
–destinationType javax.jms.Queue –ccdtUrl "http://gorillaaction:9080/ccdt/
amqclchl.tab" –ccdtQmgrName QM3}
spec3(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234988)
```

deleteWMQActivationSpec command

Use this command to delete a WebSphere MQ messaging provider activation specification at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the deleteWMQActivationSpec command to delete a WebSphere MQ messaging provider activation specification defined at the scope at which the command is issued.

Target object

A WebSphere MQ messaging provider activation specification at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQActivationSpecs("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
unwantedSpec(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)

wsadmin>AdminTask.deleteWMQActivationSpec("unwantedSpec(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#J2CActivationSpec_1098737234986)")
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQActivationSpecs
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
unwantedSpec(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)

wsadmin>$AdminTask deleteWMQActivationSpec
unwantedSpec(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

listWMQActivationSpecs command

Use the listWMQActivationSpecs command to list WebSphere MQ messaging provider activation specifications.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the listWMQActivationSpecs command to list all of the WebSphere MQ messaging provider activation specifications defined at the scope at which the command is issued.

Target object

WebSphere MQ messaging provider activation specifications at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQActivationSpecs("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQActivationSpecs
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

modifyWMQActivationSpec command

Use the modifyWMQActivationSpec command to change certain parameters of a WebSphere MQ messaging provider activation specification.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the modifyWMQActivationSpec command to modify a WebSphere MQ messaging provider activation specification defined at the scope at which the command is issued.

Note: You cannot change the type of an activation specification. For example, you cannot create an activation specification where you enter all the configuration information manually and then modify it to use a CCDT.

For a CCDT based activation specification, you cannot modify of the following parameters:

- qmgrName
- qmgrHostname
- qmgrPortNumber
- qmgrSrvconnChannel
- transportChain
- wmqTransportType

For a generic activation specification, you cannot modify any of the following parameters:

- ccdtUrl
- ccdtQmgrName

Target object

A WebSphere MQ messaging provider activation specification at the specific scope.

Required parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider activation specification.

Optional parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider activation specification.

Note the behavior of this command on the **-customProperties** parameter.

-customProperties

This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider activation specification implementation. Typically, custom properties are used to set attributes of the activation specification which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

New name/value pairs are added to the existing set of custom properties using the following rules:

- If the existing set of properties does not contain a property with the same name as that supplied as part of a modify command, the supplied property is added to the set of custom properties, unless the custom property has no value specified, when it is disregarded.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, and the modify command also specifies a value for the property, the existing value is replaced by the supplied value.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, but the modify command does not specify a value for the property, the property with the same name is deleted from the existing set of custom properties.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQActivationSpecs("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
```

```
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)


wsadmin>AdminTask.modifyWMQActivationSpec("spec1(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#J2CActivationSpec_1098737234986)",
["-destinationJndiName jms/topics/t5 –destinationType javax.jms.Topic"])
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQActivationSpecs
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)


wsadmin>$AdminTask modifyWMQActivationSpec
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
{-destinationJndiName jms/topics/t5 –destinationType javax.jms.Topic}
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
```

showWMQActivationSpec command

Use the showWMQActivationSpec command to display information about a specific WebSphere MQ messaging provider activation specification.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the showWMQActivationSpec command to display all the parameters, and their values, associated with a particular WebSphere MQ messaging provider activation specification.

Target object

A WebSphere MQ messaging provider activation specification at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQActivationSpecs("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)

wsadmin>AdminTask.showWMQActivationSpec("spec1(cells/9994GKCNode01Cell/nodes/
9994GKCNode01|resources.xml#J2CActivationSpec_1098737234986)")
{cleanupLevel=SAFE, useConnectionPooling=true, port=1414, maxPoolDepth=10,
channel=channel1, transportType=CLIENT, subscriptionStore=MIGRATE,
messageSelection=CLIENT, cleanupInterval=3600000,
brokerCCSubQueue=SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE, name=spec1, CCSID=819,
useJNDI=true, hostName=localhost, rescanInterval=5000, headerCompression=NONE,
brokerCCDurSubQueue=SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE, queueManager=QMGR1,
messageCompression=NONE, startTimeout=10000,  destinationJndiName=jms/q1,
poolTimeout=300000, sslType=NONE,
destinationType=javax.jms.Queue, brokerSubQueue=SYSTEM.JMS.ND.SUBSCRIBER.QUEUE,
sslResetCount=0, brokerControlQueue=SYSTEM.BROKER.CONTROL.QUEUE,
statusRefreshInterval=60000, cloneSupport=DISABLED, jndiName=jms/as1,
sparseSubscriptions=FALSE, authenticationAlias=null, failIfQuiesce=true,
description=, brokerVersion=1}
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQActivationSpecs
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)

wsadmin>$AdminTask showWMQActivationSpec
spec1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
J2CActivationSpec_1098737234986)
{cleanupLevel=SAFE, useConnectionPooling=true, port=1414, maxPoolDepth=10,
channel=channel1, transportType=CLIENT, subscriptionStore=MIGRATE,
messageSelection=CLIENT, cleanupInterval=3600000,
brokerCCSubQueue=SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE, name=spec1, CCSID=819,
useJNDI=true, hostName=localhost, rescanInterval=5000, headerCompression=NONE,
brokerCCDurSubQueue=SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE, queueManager=QMGR1,
messageCompression=NONE, startTimeout=10000,  destinationJndiName=jms/q1,
poolTimeout=300000, sslType=NONE,
destinationType=javax.jms.Queue, brokerSubQueue=SYSTEM.JMS.ND.SUBSCRIBER.QUEUE,
sslResetCount=0, brokerControlQueue=SYSTEM.BROKER.CONTROL.QUEUE,
statusRefreshInterval=60000, cloneSupport=DISABLED, jndiName=jms/as1,
sparseSubscriptions=FALSE, authenticationAlias=null, failIfQuiesce=true,
description=, brokerVersion=1}
```

createWMQConnectionFactory command

Use the createWMQConnectionFactory command to create a connection factory for the WebSphere MQ messaging provider at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the createWMQConnectionFactory command to create a WebSphere MQ messaging provider connection factory at a specific scope.

You cannot create a WebSphere MQ messaging provider connection factory under either of the following conditions:

- A WebSphere MQ messaging provider connection factory already exists with the same name, at the same scope
- The JNDI name clashes with another entry in WebSphere Application Server JNDI

Target object

The scope of the WebSphere MQ messaging provider at which the WebSphere MQ messaging provider connection factory is to be created.

Required parameters

-name
The administrative name assigned to this WebSphere MQ messaging provider connection factory.

-jndiName
The name and location used to bind this object into WebSphere Application Server JNDI.

-type
Use this parameter to determine whether a unified connection factory, a queue connection factory or a topic connection factory is to be created.

Enter one of the following values:

- `CF`
- `QCF`
- `TCF`

`CF` is the default value.

If you specify `QCF`, you cannot specify any of the following parameters:

- **-brokerCtrlQueue**
- **-brokerSubQueue**
- **-brokerCCSubQueue**
- **-brokerVersion**

- **-brokerPubQueue**
- **-tempTopicPrefix**
- **-pubAckWindow**
- **-subStore**
- **-stateRefreshHint**
- **-cleanupLevel**
- **-sparesSubs**
- **-wildcardFormat**
- **-brokerQmgr**
- **-clonedSubs**
- **-msgSelection**

If you specify `TCF`, you cannot specify any of the following parameters:

- **-msgRetention**
- **-rescanInterval**
- **-tempQueuePrefix**
- **-modelQueue**
- **-replyWithRFH2**

Optional parameters

-description
An administrative description assigned to the connection factory.

-ccdtUrl
A URL to a client channel definition table to use, for this connection factory, when contacting WebSphere MQ.

Use this parameter to create a ccdtURL connection factory

Do not specify this parameter in conjunction with the following parameters: **-qmgrName**, **-qmgrType**, **-qmgrHostname**, **-qmgrPortNumber**, **-qmgrSvrconnChannel**, **-transportChain** or **-localAddress**.

-ccdtQmgrName
A queue manager name, used to select one or more entries from a client channel definition table.

Do not specify this parameter in conjunction with the following parameters: **-qmgrName**, **-qmgrType**, **-qmgrHostname**, **-qmgrPortNumber**, **-qmgrSvrconnChannel**, **-transportChain**, or **-localAddress**.

-qmgrName
The name of the queue manager to use, for this connection factory, when contacting WebSphere MQ.

Use this parameter to create a generic connection factory.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-wmqTransportType
This parameter determines the way in which a connection is established to WebSphere MQ for this connection factory.

Use this parameter to create a generic connection factory.

Enter one of the following values:

- `BINDINGS`
- `BINDINGS_THEN_CLIENT`
- `CLIENT`

BINDINGS_THEN_CLIENT is the default value.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

For more information about configuring a transport type of *bindings then client* or *bindings*, refer to "Configuring the WebSphere MQ messaging provider with native libraries information" on page 1857and "Sizing the thread pools used by the WebSphere MQ messaging provider" on page 1857.

-qmgrHostname
The host name to use, for this connection factory, when attempting a client mode connection to WebSphere MQ. It must be a valid TCP/IP hostname or IPv4 or IPv6 address.

The default value is the local host.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-qmgrPortNumber
The port number to use, for this connection factory, when attempting a client mode connection to WebSphere MQ.

Enter an integer value in the range 1 – 65536 (inclusive).

The default value is 1414.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-containerAuthAlias
The container-managed authentication alias, defined to the cell, from which security credentials are used to establish a connection to WebSphere MQ.

-componentAuthAlias
The component-managed authentication alias, defined to the cell, from which security credentials are used to establish a connection to WebSphere MQ.

-clientId
The client identifier used for connections started using this connection factory.

-providerVersion
This parameter determines the minimum version, and capabilities of the queue manager.

Enter values in one of the following formats:

- *n*
- *n.n*
- *n.n.n*
- *n.n.n.n*

where *n* is an integer greater than or equal to zero.

-sslCrl
This parameter specifies a list of LDAP servers that are used to provide certificate revocation information if this connection factory establishes an SSL based connection to WebSphere MQ.

-sslResetCount
This parameter is used when the connection factory establishes an SSL connection to the queue manager. This parameter determines how many bytes to transfer before resetting the symmetric encryption key that is used for the SSL session.

Enter a value in the range 0 through 999,999,999.

The default value is 0.

-sslPeerName
This parameter is used when the connection factory establishes an SSL connection to the queue manager. The value is compared with the distinguished name present in the peer's certificate.

-rcvExit
A comma-separated list of receive exit class names.

-rcvExitInitData
Initialization data to pass to the receive exit.

Do not specify this parameter unless you specify the **-rcvExit** parameter.

-sendExit
A comma-separated list of send exit class names.

-sendExitInitData
Initialization data to pass to the send exit.

Do not specify this parameter unless you specify the **-sendExit** parameter.

-secExit
A security exit class name.

-secExitInitData
Initialization data to pass to the security exit.

Do not specify this parameter unless you specify the **-secExit** parameter.

-compressHeaders
This parameter determines if message headers are compressed.

Enter one of the following values:

- NONE
- SYSTEM

The default value is NONE.

-compressPayload
This parameter determines if message payloads are compressed.

Enter one of the following values:

- NONE
- RLE
- ZLIBFAST
- ZLIBHIGH

The default value is NONE.

-msgRetention
This parameter determines if the connection consumer keeps unwanted messages on the input queue.

Enter one of the following values:

- YES
- NO

where YES specifies that the connection consumer keeps unwanted messages on the input queue.

and NO specifies that the messages are disposed of according to their disposition options.

The default value is YES.

-pollingInterval
This property is applicable in the client container only.

If each message listener within a session has no suitable message on its queue, this parameter is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue. If it frequently happens that no suitable message is available for any of the message listeners in a session, consider increasing the value of this parameter.
The default value is `5000`.

-rescanInterval
When a message consumer in the point-to-point domain uses a message selector to select which messages it wants to receive, the JMS client searches the WebSphere MQ queue for suitable messages in the sequence determined by the MsgDeliverySequence attribute of the queue. When the client finds a suitable message and delivers it to the consumer, the client resumes the search for the next suitable message from its current position in the queue. The client continues to search the queue in this way until it reaches the end of the queue, or until the interval of time in milliseconds, as determined by the value of this **-rescanInterval** parameter has expired. In each case, the client returns to the beginning of the queue to continue its search, and a new time interval commences

This parameter must be a positive integer value.

The default value is `5000`.

-ccsid
The coded character set identifier to be used on connections.

The value of this parameter must be a positive integer.

The default value is `819`.

-failIfQuiescing
This parameter determines the behavior of certain calls to the queue manager when the queue manager is put into quiescing state.

The value of this parameter must be `True` or `False`.

`True` specifies that calls to certain methods fail if the queue manager is in a quiescing state. If an application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop.

`False` specifies that no methods fail if the queue manager is in a quiescing state. If you specify this value, an application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager, and therefore prevent the queue manager from stopping.

The default value is `True`.

-brokerCtrlQueue
The name of the broker control queue to use if this connection factory is to subscribe to a topic.

The default value is `SYSTEM.BROKER.CONTROL.QUEUE`.

-brokerSubQueue
The name of the queue to use for obtaining subscription messages if this connection factory is to subscribe to a topic.

The default value is `SYSTEM.JMS.ND.SUBSCRIBER.QUEUE`.

-brokerCCSubQueue
The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer.

The default value is `SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE`.

-brokerVersion
The value of this parameter determines the level of functionality required for publish/subscribe operations.

Valid values are 1 and 2.

The default value is 1.

-msgSelection
This parameter determines where message selection occurs.

Valid values are CLIENT and BROKER.

The default value is CLIENT.

-subStore
This parameter determines where WebSphere MQ messaging provider stores persistent data relating to active subscriptions.

Valid values are MIGRATE, QUEUE and BROKER.

The default value is MIGRATE.

-stateRefreshHint
The interval, in milliseconds, between refreshes of the long running transaction that detects when a subscriber loses its connection to the queue manager. This parameter is relevant only if **-subStore** parameter has the value QUEUE.

The value of this parameter must be a positive integer.

The default value is 60,000.

-cleanupLevel
The cleanup level for BROKER or MIGRATE subscription stores

Valid values are SAFE, ASPROP, NONE and STRONG.

The default value is SAFE.

-cleanupInterval
The interval between background executions of the publish/subscribe cleanup utility.

The value of this parameter must be a positive integer.

The default value is 3,600,000.

-wildcardFormat
This parameter determines which sets of characters are interpreted as topic wildcards.

Valid values are Topic or Char.

The default value is Char.

-sparseSubs
This parameter controls the message retrieval policy of a TopicSubscriber object.

The value of this parameter must be True or False

The default value is False.

-brokerQmgr
The name of the queue manager that is running the broker, if it is not the same as the queue manager to which the connection factory connects.

There is no default value.

-clonedSubs
This parameter determines whether two or more instances of the same durable topic subscriber can run simultaneously

The value of this parameter must be ENABLED or DISABLED

The default value is ENABLED.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider connection factory implementation. Typically, custom properties are used to set attributes of the connection factory which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

-qmgrSvrconChannel
The SVRCONN channel to use when connecting to WebSphere MQ.

Use this parameter to create an **explicitly defined** connection factory.

The default value is `SYSTEM.DEF.SVRCONN`.

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-support2PCProtocol
This parameter determines if the connection factory acts as a resource that is capable of participation in distributed two phase commit processing.

The value of this parameter must be `True` or `False`.

The default value `True` specifies that the connection factory acts as a resource that is capable of participation in distributed two phase commit processing.

-modelQueue
The name of the WebSphere MQ model queue whose definition is used as a basis when creating JMS temporary destinations.

The default value is `SYSTEM.DEFAULT.MODEL.QUEUE`.

-tempQueuePrefix
The prefix to apply to WebSphere MQ temporary queues that are used to represent JMS temporary queue type destinations.

There is no default value.

-tempTopicPrefix
The prefix to apply to the names generated for temporary topics. This parameter is only valid for connection factories or topic connection factories.

There is no default value.

-replyWithRFH2
This parameter determines whether, when replying to a message, a RFH version 2 header is included in the reply message.

The value of this parameter must be `ALWAYS` or `AS_REPLY_DEST`

The default value is `AS_REPLY_DEST`.

-brokerPubQueue
The name of the queue to send publication messages to when using queue based brokering.

The default value is `SYSTEM.BROKER.DEFAULT.STREAM`.

-pubAckInterval
The number of publications to send to a queue based broker before sending a publication that solicits an acknowledgement.

The value of this parameter must be a positive integer greater than zero.

The default value is `25`.

-sslType
This parameter determines the configuration, if any, to use when applying SSL encryption to the network connection to the queue manager.

The value of this parameter must be `CENTRAL`, `SPECIFIC` or `NONE`

The **-sslConfiguration** parameter is not valid unless this parameter is set to `SPECIFIC`.

The default value is `NONE`.

The **sslConfiguration** parameter is not valid unless this parameter is set to the value `SPECIFIC`.

-sslConfiguration
The name of the SSL configuration to use when using SSL to secure network connections to the queue manager.

Do not specify this parameter unless the parameter **-sslType** is assigned the value `SPECIFIC`.

The value of this parameter must correspond to an SSL configuration.

-localAddress
This parameter specifies either or both of the following:

- the local network interface
- the local port, or range of local ports

Do not specify this parameter in conjunction with the following parameters: **-ccdtUrl** or **-ccdtQmgrName**.

-mappingAlias
The JAAS mapping alias used when determining which security credentials to use when establishing a connection to WebSphere MQ.

The default value is DefaultPrincipleMapping.

-xaRecoveryAuthAlias
The authentication alias from which credentials are taken and used to connect to WebSphere MQ for XA recovery.

There is no default value.

Minimal connection factory definition

The following example creates an connection factory, specifying the minimum number of parameters. Due to the default values assumed for the unspecified parameters, applications using this connection factory expect to be co-located with a queue manager installed on the same node.

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQConnectionFactory
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name cf1 –jndiName "jms/cf/cf1" –type CF}
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1205322636000)
```

Explicitly defined connection factory

The following example creates an connection factory for which the user must specify and maintain all the parameters used for establishing a connection to WebSphere MQ.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQConnectionFactory("9994GKCNode01(cells/
```

```
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)", ["-name cf2
 –jndiName 'jms/cf/cf2' –type CF –description 'Must remember to keep each
of these connection factories in sync with the WebSphere MQ queue manager
to which they refer' –qmgrName QM1 –qmgrHostname 192.168.0.22 –qmgrPort 1415
 –qmgrSvrconnChannel QM1.SVRCONN"])
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263601)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQConnectionFactory
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name cf2 –jndiName "jms/cf/cf2" –type CF –description "Must remember to
keep each of these connection factories in sync with the WebSphere MQ queue
manager to which they refer" –qmgrName QM1 –qmgrHostname 192.168.0.22
–qmgrPort 1415 –qmgrSvrconnChannel QM1.SVRCONN}
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263601)
```

Connection factory definition specifying a CCDT

The following example creates an connection factory that uses a CCDT to locate the queue manager to connect to.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQConnectionFactory("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)", ["-name cf3 –jndiName
 'jms/cf/cf3' –type CF –ccdtUrl 'http://gorillaaction:9080/ccdt/amqclchl.tab'
 –ccdtQmgrName QM3"])
cf3(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263606)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQConnectionFactory
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name cf3 –jndiName "jms/cf/cf3" –type CF –ccdtUrl
"http://gorillaaction:9080/ccdt/amqclchl.tab" –ccdtQmgrName QM3}
cf3(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_120532263606)
```

deleteWMQConnectionFactory command

Use the deleteWMQConnectionFactory command to delete a WebSphere MQ messaging provider connection factory at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the deleteWMQConnectionFactory command to delete a WebSphere MQ messaging provider connection factory defined at the scope at which the command is issued.

Target object

A WebSphere MQ messaging provider connection factory at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQConnectionFactories("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)")
unwantedCF(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>AdminTask.deleteWMQConnectionFactory("unwantedCF(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#MQConnectionFactory_1098737234986)")
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQConnectionFactories
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
unwantedCF(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>$AdminTask deleteWMQConnectionFactory
unwantedCF(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
```

listWMQConnectionFactories command

Use the listWMQConnectionFactories command to list WebSphere MQ messaging provider connection factories.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the listWMQConnectionFactories command to list all of the WebSphere MQ messaging provider connection factories defined at the scope at which the command is issued.

Target object

WebSphere MQ messaging provider connection factories at the specific scope.

Required parameters

None.

Optional parameters

-type

Use this parameter to determine which type of connection factory is listed. If it is omitted all connection factories are shown at the appropriate scope.

Valid values are:

- `CF` to list only common connection factories
- `QCF` to list only queue connection factories
- `TCF` to list only topic connection factories

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQConnectionFactories("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)")
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQConnectionFactories
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
cf2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
```

modifyWMQConnectionFactory command

Use the modifyWMQConnectionFactory command to change certain parameters of a WebSphere MQ messaging provider connection factory.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the modifyWMQConnectionFactory command to modify a WebSphere MQ messaging provider connection factory defined at the scope at which the command is issued.

Note: When modifying a WebSphere MQ messaging provider connection factory, there is an interaction between the **mappingAlias** and **containerAuthAlias** parameters. This interaction occurs if the **containerAuthAlias** parameter is specified but the **mappingAlias** is not specified. In this situation, the **mappingAlias** parameter is automatically set to the value DefaultPrincipleMapping.

Target object

A WebSphere MQ messaging provider connection factory at the specific scope.

Required parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider connection factory.

Optional parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider connection factory.

Note the behavior of this command on the **-customProperties** parameter.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider connection factory implementation. Typically, custom properties are used to set attributes of the connection factory which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

New name/value pairs are added to the existing set of custom properties using the following rules:

- If the existing set of properties does not contain a property with the same name as that supplied as part of a modify command, the supplied property is added to the set of custom properties, unless the custom property has no value specified, when it is disregarded.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, and the modify command also specifies a value for the property, the existing value is replaced by the supplied value.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command but the modify command does not specify a value for the property, the property with the same name is deleted from the existing set of custom properties.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQConnectionFactories("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)")
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>AdminTask.modifyWMQConnectionFactory("cf1(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#MQConnectionFactory_1098737234986)", ["-description
 'My new description'"])
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQConnectionFactories
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>$AdminTask modifyWMQConnectionFactory
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986) {-description "My new description"}
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
```

showWMQConnectionFactory command

Use the showWMQConnectionFactory command to display information about a specific WebSphere MQ messaging provider connection factory.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the showWMQConnectionFactory command to display all the parameters, and their values, associated with a particular WebSphere MQ messaging provider connection factory.

Target object

A WebSphere MQ messaging provider connection factory at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQConnectionFactories("9994GKCNode01(cells/
9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)")
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>AdminTask.showWMQConnectionFactory("cf1(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#MQConnectionFactory_1098737234986)")
{ name=cf1 jndiName=jms/cf/cf1 description= qmgrName=QMGR1
qmgrSvrconnChannel=TO.QMGR1 qmgrHostname=localhost qmgrPortNumber=1414
wmqTransportType=bindingsThenClient authAlias= clientId="Bob's Magic Client"
providerVersion= sslCrl= sslResetCount= sslPeerName= rcvExit=
rcvExitInitData= sendExit= sendExitInitData= secExit= secExitInitData=
compressHeaders=NONE compressPayload=NONE msgRetention=true
pollingInterval=5000 rescanInterval=5000 maxBatchSize=10 ccsid=819
failIfQuiescing=true brokerCtrlQueue=
brokerSubQueue=SYSTEM.BROKER.CONTROL.QUEUE
brokerCCSubQueue=SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
brokerVersion=1 msgSelection=CLIENT subStore=MIGRATE stateRefreshHint=60000
cleanupInterval=360000000 cleanupLevel=SAFE wildcardFormat=CHAR
sparseSubs=false brokerQmgr= clonedSubs=true}
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQConnectionFactories
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)

wsadmin>$AdminTask showWMQConnectionFactory
cf1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQConnectionFactory_1098737234986)
{ name=cf1 jndiName=jms/cf/cf1 description= qmgrName=QMGR1
```

```
qmgrSvrconnChannel=TO.QMGR1 qmgrHostname=localhost qmgrPortNumber=1414
wmqTransportType=bindingsThenClient authAlias= clientId="Bob's Magic Client"
providerVersion= sslCrl= sslResetCount= sslPeerName= rcvExit=
rcvExitInitData= sendExit= sendExitInitData= secExit= secExitInitData=
compressHeaders=NONE compressPayload=NONE msgRetention=true
pollingInterval=5000 rescanInterval=5000 maxBatchSize=10 ccsid=819
failIfQuiescing=true brokerCtrlQueue=
brokerSubQueue=SYSTEM.BROKER.CONTROL.QUEUE
brokerCCSubQueue=SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
brokerVersion=1 msgSelection=CLIENT subStore=MIGRATE stateRefreshHint=60000
cleanupInterval=360000000 cleanupLevel=SAFE wildcardFormat=CHAR
sparseSubs=false brokerQmgr= clonedSubs=true}
```

createWMQTopic command

Use the createWMQTopic command to create a JMS topic destination for the WebSphere MQ messaging provider at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the createWMQTopic command to create a WebSphere MQ messaging provider topic type destination at a specific scope.

You cannot create a WebSphere MQ messaging provider topic type destination under either of the following conditions:

- A WebSphere MQ messaging provider topic type destination already exists with the same name, at the same scope.
- The JNDI name clashes with another entry in WebSphere Application Server JNDI.

Target object

The scope of the WebSphere MQ messaging provider at which the WebSphere MQ messaging provider topic type destination is to be created.

Required parameters

-name
The administrative name assigned to this WebSphere MQ messaging provider topic type destination.

-jndiName
The name used to bind this object into WebSphere Application Server JNDI.

-topicName
The name of the WebSphere MQ topic where publications are received from, or sent to, when this destination definition is used.

Optional parameters

-description
An administrative description assigned to the topic type destination.

-persistence
This parameter determines the level of persistence used to store messages sent to this destination.

Enter one of the following case-sensitive values:

- `APP`
- `TDEF`
- `PERS`
- `NON`
- `HIGHT`

`APP` is the default value.

-priority
The priority level to assign to messages sent to this destination.

Enter one of the following case-sensitive values:

- `APP`
- `TDEF`

or enter a positive integer in the range 0 to 9 (inclusive).

`APP` is the default value.

-expiry
The length of time after which messages that are sent to this destination expire and are dealt with according to their disposition options.

Enter one of the following case-sensitive values:

- `APP`
- `UNLIM`

or enter any positive integer.

`APP` is the default value.

-ccsid
The coded character set identifier.

The value of this parameter must be a positive integer.

The default value is `1208`.

-useNativeEncoding
This parameter specifies whether to use native encoding or not. It can take a value `true` or `false`.

If it is set to `true`, the values of the **-integerEncoding**, **-decimalEncoding** and **-floatingPointEncoding** attributes are ignored.

If it is set to `false`, the encoding is specified by the **-integerEncoding**, **-decimalEncoding** and **-floatingPointEncoding** attributes.

-integerEncoding
The integer encoding setting for this queue.

Enter one of the following case-sensitive values: `Normal`, `Reversed`.

`Normal` is the default value.

-decimalEncoding
The decimal encoding setting for this queue.

Enter one of the following case-sensitive values: `Normal`, `Reversed`.

`Normal` is the default value.

-floatingPointEncoding
The floating point encoding setting for this queue.

Enter one of the following case-sensitive values: `IEEENormal`, `IEEEReversed`, `z/OS`

`IEEENormal` is the default value.

-useRFH2
This parameter determines whether an RFH version 2 header is appended to messages sent to this destination.

Enter one of the following case-sensitive values: `true` or `false`.

`true` is the default value.

-sendAsync
This parameter determines whether messages can be sent to this destination without queue manager acknowledging that they have arrived.

Enter one of the following case-sensitive values: `YES`, `NO` or `TDEF`.

`YES` is the default value.

-readAhead
This parameter determines whether messages for non-persistent consumers can be read ahead and cached.

Enter one of the following case-sensitive values: `YES`, `NO` or `TDEF`.

`YES` is the default value.

-readAheadClose
This property determines the behavior that occurs when closing a message consumer that is receiving messages asynchronously using a message listener from a destination that has the **readAhead** parameter set to `True`.

When a value of `deliverAll` is specified, all read-ahead messages are delivered before closing the consumer.

When a value of `deliverCurrent` is specified, only in-progress messages are delivered before closing the consumer.

`deliverCurrent` is the default value.

-wildcardFormat
This parameter determines which sets of characters are interpreted as topic wildcards.

Valid values are `Topic` or `Char`.

`Char` is the default value.

-brokerDurSubQueue
The name of the queue, defined to the queue manager, from which a connection consumer receives non-durable subscription messages.

The value of this parameter must be a valid queue name or left blank

The default value is `SYSTEM.JMS.D.SUBSCRIBER.QUEUE`.

-brokerCCDurSubQueue
The name of the queue, defined to the queue manager, from which a connection consumer receives durable subscription messages.

The value of this parameter must be a valid queue name or left blank

The default value is `SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE`.

-brokerPubQueue
The name of the queue, defined to the queue manager, to which publication messages are sent.

The value of this parameter must be a valid queue name or left blank

The default value is `SYSTEM.BROKER.DEFAULT.STREAM`.

-brokerPubQmgr
The name of the queue manager on which the broker is running.

The value of this parameter must be a valid queue manager name or left blank

There is no default value.

-brokerVersion
This parameter determines the level of functionality required for publish/subscribe operations.

The value of this parameter must `V1` or `V2`.

The default value is `V1`.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider topic type destination implementation. Typically, custom properties are used to set attributes of the topic type destination that are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of **-name** and **-value** parameters.

The following example creates a topic definition by specifying the minimum number of parameters.

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
 9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.createWMQTopic("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)", ["-name T1 –jndiName jms/topic/t1
 –topicName woodland/creatures/badger"])
T1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
 9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask createWMQTopic
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name T1 –jndiName jms/topic/t1 –topicName woodland/creatures/badger}
T1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
```

deleteWMQTopic command

Use this command to delete a WebSphere MQ messaging provider topic at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the deleteWMQTopic command to delete a WebSphere MQ messaging provider topic defined at the scope at which the command is issued.

Target object

A WebSphere MQ messaging provider topic at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQTopics("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
unwantedTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)

wsadmin>$AdminTask deleteWMQTopic
unwantedTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQTopics
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
unwantedTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)

wsadmin>$AdminTask deleteWMQTopic
unwantedTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
```

listWMQTopics command

Use the listWMQTopics command to list WebSphere MQ messaging provider topics.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the listWMQTopics command to list all of the WebSphere MQ messaging provider topics defined at the scope at which the command is issued.

Target object

WebSphere MQ messaging provider topics at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQTopics("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
aaaTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
```

```
bbbTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234987)
cccTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234988)
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQTopics
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
aaaTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
bbbTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234987)
cccTopic(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234988)
```

modifyWMQTopic command

Use the modifyWMQTopic command to change certain parameters of a WebSphere MQ messaging provider topic.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the modifyWMQTopic command to modify a WebSphere MQ messaging provider topic defined at the scope at which the command is issued.

Note: You cannot change the name of an topic.

Target object

A WebSphere MQ messaging provider topic at the specific scope.

Required parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider topic.

Optional parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider topic.

Note the behavior of this command on the **-customProperties** parameter.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider topic implementation. Typically, custom properties are used to set attributes of the topic which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

New name/value pairs are added to the existing set of custom properties using the following rules:

- If the existing set of properties does not contain a property with the same name as that supplied as part of a modify command, the supplied property is added to the set of custom properties, unless the custom property has no value specified, when it is disregarded.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, and the modify command also specifies a value for the property, the existing value is replaced by the supplied value.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, but the modify command does not specify a value for the property, the property with the same name is deleted from the existing set of custom properties.

Example

- Using Jython:

```
wsadmin>AdminTask.modifyWMQTopic("t1(cells/L3A3316Node04Cell/
nodes/L3A3316Node05|resources.xml#MQTopic_1204538835312)", ["-priority 7"])
t1(cells/L3A3316Node04Cell/nodes/L3A3316Node05|resources.xml#
MQTopic_1204538835312)
```

- Using Jacl:

```
wsadmin>$AdminTask modifyWMQTopic
t1(cells/L3A3316Node04Cell/nodes/L3A3316Node05|resources.xml#
MQTopic_1204538835312) {-priority 7}
t1(cells/L3A3316Node04Cell/nodes/L3A3316Node05|resources.xml#
MQTopic_1204538835312)
```

showWMQTopic command

Use the showWMQTopic command to display information about a specific WebSphere MQ messaging provider topic.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the showWMQTopic command to display all the parameters, and their values, associated with a particular WebSphere MQ messaging provider topic.

Target object

A WebSphere MQ messaging provider topic at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQTopics("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
topic1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)

wsadmin>AdminTask.showWMQTopic("topic1(cells/9994GKCNode01Cell/nodes/
9994GKCNode01|resources.xml#MQTopic_1098737234986)")
{specifiedExpiry=0, priority=APPLICATION_DEFINED, decimalEncoding=Normal,
baseTopicName=topic1, name=topic1, readAhead=NO, brokerPubQmgr=null,
readAheadClose=DELIVERCURRENT, brokerPubQueue=SYSTEM.BROKER.DEFAULT.STREAM,
ccsid=1208, integerEncoding=Normal, useNativeEncoding=true,
wildcardFormat=charWildcards, expiry=APP, specifiedPriority=0,
jndiName=jms/t1, sendAsync=NO,
brokerDurSubQueue=SYSTEM.JMS.D.SUBSCRIBER.QUEUE, brokerCCDurSubQueue=null,
description=null, targetClient=JMS, floatingPointEncoding=IEEENormal,
persistence=APP, brokerVersion=V1}
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQTopics
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
topic1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)

wsadmin>$AdminTask showWMQTopic
topic1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQTopic_1098737234986)
{specifiedExpiry=0, priority=APPLICATION_DEFINED, decimalEncoding=Normal,
baseTopicName=topic1, name=topic1, readAhead=NO, brokerPubQmgr=null,
readAheadClose=DELIVERCURRENT, brokerPubQueue=SYSTEM.BROKER.DEFAULT.STREAM,
ccsid=1208, integerEncoding=Normal, useNativeEncoding=true,
wildcardFormat=charWildcards, expiry=APP, specifiedPriority=0,
jndiName=jms/t1, sendAsync=NO,
brokerDurSubQueue=SYSTEM.JMS.D.SUBSCRIBER.QUEUE, brokerCCDurSubQueue=null,
description=null, targetClient=JMS, floatingPointEncoding=IEEENormal,
persistence=APP, brokerVersion=V1}
```

manageWMQ command

Use the manageWMQ command to manage the settings of the WebSphere MQ resource adapter installed in a node.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the manageWMQ command to manage the settings associated with the WebSphere MQ resource adapter installed in a node.

The manageWMQ command allows you to manage the native library and query the meta-data of the WebSphere MQ resource adapter on each node.

Target object

A WebSphere MQ resource adapter on a specific node.

Required parameters

None.

Optional parameters

-nativePath
This parameter specifies the path to the WebSphere MQ messaging provider native libraries that are used by the WebSphere MQ resource adapter to establish a bindings mode connection to the queue manager.

-query
This parameter provides information about the WebSphere MQ messaging provider. This can be either the WebSphere MQ resource adapter installed into WebSphere Application Server or a WebSphere MQ resource adapter present on the file system of the node.

Example

The following example shows how to enable inbound JCA message delivery on the z/OS platform.

- Using Jython:

```
wsadmin>AdminTask.manageWMQ("WebSphere MQ Resource Adapter
(cells/L3A3316Node04Cell/nodes/L3A3316Node05/servers/server1|resources.xml#
J2CResourceAdapter_1201601803796)", ["-enableInbound true"])
```

- Using Jacl:

```
wsadmin>$AdminTask manageWMQ "WebSphere MQ Resource Adapter
(cells/L3A3316Node04Cell/nodes/L3A3316Node05/servers/server1|resources.xml#
J2CResourceAdapter_1201601803796)" {-enableInbound true}
```

migrateWMQMLP command

Use the migrateWMQMLP command to migrate a WebSphere MQ message listener port definition to an activation specification definition.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the migrateWMQMLP command to migrate a WebSphere MQ message listener port definition to an activation specification definition. After the activation specification has been created, you can delete the listener port.

Target object

The message listener port to be migrated.

Required parameters

-asName
The name of the activation specification to be created.

-asJNDIName
The JNDI name of the activation specification to be created.

-asScope
The type of scope at which to create the activation specification (server, node, cluster or cell). Note that the cluster option is only supported when the server that contains the message listener port is part of a cluster. If not specified this defaults to server. The scopes specified are relative to the message listener port, so node is the node of the server that contains the message listener port.

Optional parameters

None.

Example

The following example shows how to migrate a message listener port to an activation specification.

- Using Jython:

```
wsadmin>AdminConfig.list("ListenerPort")
lp1(cells/L3A3316Node09Cell/nodes/L3A3316Node10/servers/server1|
server.xml#ListenerPort_1211265363796)

wsadmin>AdminTask.migrateWMQMLP("lp1(cells/L3A3316Node09Cell/nodes/
L3A3316Node10/servers/server1|server.xml#ListenerPort_1211265363796)",
 ["-asName migratedFromLP -asJNDIName jms/as1 -asScope node"])
migratedFromLP(cells/L3A3316Node09Cell/nodes/L3A3316Node10|
resources.xml#J2CActivationSpec_1211265679078)
```

- Using Jacl:

```
wsadmin>$AdminConfig list ListenerPort
lp1(cells/L3A3316Node09Cell/nodes/L3A3316Node10/servers/server1|
server.xml#ListenerPort_1211265363796)

wsadmin>$AdminTask migrateWMQMLP
lp1(cells/L3A3316Node09Cell/nodes/L3A3316Node10/servers/server1|
server.xml#ListenerPort_1211265363796)
{-asName migratedFromLP -asJNDIName jms/as1 -asScope node}
migratedFromLP(cells/L3A3316Node09Cell/nodes/L3A3316Node10|
resources.xml#J2CActivationSpec_1211265679078)
```

createWMQQueue command

Use the createWMQQueue command to create a queue type destination for the WebSphere MQ messaging provider at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the createWMQQueue command to create a WebSphere MQ messaging provider queue type destination at a specific scope.

You cannot create a WebSphere MQ messaging provider queue type destination under either of the following conditions:

- A WebSphere MQ messaging provider queue type destination already exists with the same name, at the same scope.
- The JNDI name clashes with another entry in WebSphere Application Server JNDI.

Target object

The scope of the WebSphere MQ messaging provider at which the WebSphere MQ messaging provider queue type destination is to be created.

Required parameters

-name
The administrative name assigned to this WebSphere MQ messaging provider queue type destination.

-jndiName
The name used to bind this object into WebSphere Application Server JNDI.

Optional parameters

-description
An administrative description assigned to the queue type destination.

-queueName
The name of the WebSphere MQ queue to use to store messages for the WebSphere MQ messaging provider queue type destination definition.

-qmgr
The queue manager on which the WebSphere MQ queue resides

-persistence
This parameter determines the level of persistence used to store messages sent to this destination.

Enter one of the following case-sensitive values:

- `APP`
- `QDEF`
- `PERS`
- `NON`
- `HIGH`

`APP` is the default value.

-priority
The priority level to assign to messages sent to this destination.

Enter one of the following case-sensitive values:

- `APP`
- `QDEF`

or enter a positive integer in the range 0 to 9 (inclusive).

`APP` is the default value.

-expiry
The length of time after which messages, sent to this destination, expire and are dealt with according to their disposition options.

Enter one of the following case-sensitive values:

- `APP`
- `UNLIM`

or enter any positive integer.

`APP` is the default value.

-ccsid
The coded character set identifier.

The value of this parameter must be a positive integer.

The default value is `1208`.

-useNativeEncoding
This parameter specifies whether to use native encoding or not. It can take a value `true` or `false`.

If it is set to `true`, the values of the **-integerEncoding**, **-decimalEncoding** and **-floatingPointEncoding** attributes are ignored.

If it is set to `false`, the encoding is specified by the **-integerEncoding**, **-decimalEncoding** and **-floatingPointEncoding** attributes.

-integerEncoding
The integer encoding setting for this queue.

Enter one of the following case-sensitive values: `Normal`, `Reversed`.

`Normal` is the default value.

-decimalEncoding
The decimal encoding setting for this queue.

Enter one of the following case-sensitive values: `Normal`, `Reversed`.

`Normal` is the default value.

-floatingPointEncoding
The floating point encoding setting for this queue.

Enter one of the following case-sensitive values: `IEEENormal`, `IEEEReversed`, `z/OS`

`IEEENormal` is the default value.

-useRFH2
This parameter determines whether an RFH version 2 header is appended to messages sent to this destination.

Enter one of the following case-sensitive values: `true` or `false`.

`true` is the default value.

-sendAsync
This parameter determines whether messages can be sent to this destination without queue manager acknowledging that they have arrived.

Enter one of the following case-sensitive values: `YES`, `NO` or `QDEF`.

`YES` is the default value.

-readAhead
This parameter determines whether messages for non-persistent consumers can be read ahead and cached.

Enter one of the following case-sensitive values: YES, NO or QDEF.

YES is the default value.

-readAheadClose

Enter one of the following case-sensitive values: YES, NO or QDEF.

YES is the default value.

-customProperties

This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider queue type destination implementation. Typically, custom properties are used to set attributes of the queue type destination that are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of **-name** and **-value** parameters.

Example

The following example creates a WebSphere MQ messaging provider queue type destination.

- Using Jython:

```
wsadmin>AdminTask.createWMQQueue("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)", ["-name queue1 –jndiName jms/queues/Q1
 –queueName APP1.QUEUE1"])
queue1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)
```

- Using Jacl:

```
wsadmin>$AdminTask createWMQQueue
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
{-name queue1 –jndiName jms/queues/Q1 –queueName APP1.QUEUE1}
queue1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)
```

deleteWMQQueue command

Use this command to delete a WebSphere MQ messaging provider queue type destination at a specific scope.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the deleteWMQQueue command to delete a WebSphere MQ messaging provider queue type destination defined at the scope at which the command is issued.

Note: The WebSphere MQ messaging provider queue type destination definition is deleted from WebSphere Application Server JNDI at the specific scope. Any messages present on the underlying WebSphere MQ queue are not affected.

Target object

A WebSphere MQ messaging provider queue type destination at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQQueues("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
unwantedQueue(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>AdminTask.deleteWMQQueue("unwantedQueue(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|resources.xml#MQQueue_1098737234986)")
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQQueues
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
unwantedQueue(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>$AdminTask deleteWMQQueue
unwantedQueue(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)
```

listWMQQueues command

Use the listWMQQueues command to list WebSphere MQ messaging provider queue type destinations.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the listWMQQueues command to list all of the WebSphere MQ messaging provider queue type destinations defined at the scope at which the command is issued.

Target object

WebSphere MQ messaging provider queue type destinations at the specific scope.

Required parameters

None.

Optional parameters

-type

If this parameter it is omitted all connection factories are shown at the appropriate scope.

Enter one of the following case-sensitive values: `CF`, `QCF` or `TCF`.Enter `CF` to list only common connection factories, `QCF` to list only queue connection factories or **TCF** to list only topic connection factories.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQQueues("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQQueues
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.
```

modifyWMQQueue command

Use the modifyWMQQueue command to change certain parameters of a WebSphere MQ messaging provider queue type destination.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

After using the command, save your changes to the master configuration. For example, use the following command:

```
AdminConfig.save()
```

Purpose

Use the modifyWMQQueue command to modify a WebSphere MQ messaging provider queue type destination defined at the scope at which the command is issued.

Target object

A WebSphere MQ messaging provider queue type destination at the specific scope.

Required parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider queue type destination.

Optional parameters

The parameters for this command are identical to those used to create a WebSphere MQ messaging provider queue type destination.

Note the behavior of this command on the **-customProperties** parameter.

-customProperties
This parameter specifies custom properties to be passed to the WebSphere MQ messaging provider queue type destination implementation. Typically, custom properties are used to set attributes of the queue type destination which are not directly supported through the WebSphere administration interfaces.

-customProperties are multi-step command parameters, consisting of tuples of `-name` and `-value` parameters.

New name/value pairs are added to the existing set of custom properties using the following rules:

- If the existing set of properties does not contain a property with the same name as that supplied as part of a modify command, the supplied property is added to the set of custom properties, unless the custom property has no value specified, when it is disregarded.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, and the modify command also specifies a value for the property, the existing value is replaced by the supplied value.
- If the existing set of properties contains a property with the same name as that supplied as part of a modify command, but the modify command does not specify a value for the property, the property with the same name is deleted from the existing set of custom properties.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQQueues("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>AdminTask.modifyWMQQueue("jmsq2(cells/9994GKCNode01Cell/nodes/
9994GKCNode01|resources.xml#
MQQueue_1098737234986)", ["-ccsid 500"])
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQQueues
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>$AdminTask modifyWMQQueue
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986) {-ccsid 500}
jmsq2(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.
```

showWMQQueue command

Use the showWMQQueue command to display information about a specific WebSphere MQ messaging provider queue type destination.

To run the command, use the AdminTask object of the wsadmin scripting client.

The wsadmin scripting client is run from Qshell. For more information, see the topic “Configure Qshell to run WebSphere Application Server scripts”.

This command is valid only when it is used with WebSphere Application Server Version 7 and later application servers. Do not use it with earlier versions.

For a list of the available WebSphere MQ messaging provider administrative commands, plus a brief description of each command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('WMQAdminCommands')
```

For overview help on a given command, enter the following command at the wsadmin prompt:

```
print AdminTask.help('command_name')
```

Purpose

Use the showWMQQueue command to display all the parameters, and their values, associated with a particular WebSphere MQ messaging provider queue type destination.

Target object

A WebSphere MQ messaging provider queue type destination at the specific scope.

Required parameters

None.

Optional parameters

None.

Example

- Using Jython:

```
wsadmin>AdminConfig.getid("/Node:9994GKCNode01")
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>AdminTask.listWMQQueues("9994GKCNode01(cells/9994GKCNode01Cell/
nodes/9994GKCNode01|node.xml#Node_1)")
ncq1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>AdminTask.showWMQQueue("ncq1(cells/9994GKCNode01Cell/nodes/
9994GKCNode01|resources.xml#MQQueue_1098737234986)")
{expiry=0, priority=9, decimalEncoding=Normal, queueName=TARGETQ, name=q1,
readAhead=YES, readAheadClose=DELIVERALL,  ccsid=1208,
useNativeEncoding=true, integerEncoding=Normal, specifiedPriority=0,
jndiName=jms/q1, sendAsync=YES,  qmgr=null, description=null,
targetClient=JMS, floatingPointEncoding=IEEENormal, persistence=APP}
```

- Using Jacl:

```
wsadmin>$AdminConfig getid /Node:9994GKCNode01
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)

wsadmin>$AdminTask listWMQQueues
9994GKCNode01(cells/9994GKCNode01Cell/nodes/9994GKCNode01|node.xml#Node_1)
ncq1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)

wsadmin>$AdminTask showWMQQueue
ncq1(cells/9994GKCNode01Cell/nodes/9994GKCNode01|resources.xml#
MQQueue_1098737234986)
{expiry=0, priority=9, decimalEncoding=Normal, queueName=TARGETQ, name=q1,
readAhead=YES, readAheadClose=DELIVERALL,  ccsid=1208,
useNativeEncoding=true, integerEncoding=Normal, specifiedPriority=0,
jndiName=jms/q1, sendAsync=YES,  qmgr=null, description=null,
targetClient=JMS, floatingPointEncoding=IEEENormal, persistence=APP}
```

Mapping of administrative console panel names to command names and WebSphere MQ names

Use this table to relate the names used in the administrative console panels to the names on the commands and the names used by WebSphere MQ.

The following tables list property mappings. Not every property WebSphere Application Server command or panel property maps to a WebSphere MQ property.

Use these tables in conjunction with the WebSphere MQ information center to get more information on a particular property.

WebSphere MQ Activation Specification property mappings:

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Basic panel		

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Description	description	DESC
Queue manager	qmgrName	QMGR
Queue manager (when used with CCDT)	ccdtQmgrName	QMGR
Transport	wmqTransportType	TRAN
Hostname	qmgrHostname	HOST
Port	qmgrPortNumber	PORT
Server connection channel	qmgrSvrconnChannel	CHAN
Client ID	clientID	CID
Provider version	providerVersion	PVER
Client channel definition table URL	ccdtUrl	CCDT
Allow cloned durable subscriptions	clonedSubs	CLS
Advanced properties panel		
Compress message headers	compressHeaders	HC
Compression algorithm for message payloads	compressPayload	MC
Retain messages, even if no matching consumer is available	msgRetention	MRET
Rescan interval	rescanInterval	RINT
Maximum batch size	maxBatchSize	MBS
Coded character set identifier	ccsid	CCS
Append an RFH version 2 header to reply messages	replyWithRFH2	TCM
Fail JMS method calls if the queue manager is quiescing	failIfQuiescing	FIQ
Broker properties panel		
Broker control queue	brokerCtrlQueue	BCON
Broker durable subscriber connection consumer queue	brokerCCDurSubQueue	CCDSUB
Broker subscriber queue	brokerSubQueue	BSUB
Broker connection consumer subscription queue	brokerCCSubQueue	CCSUB
Version	brokerVersion	BVER
Specify where message selection occurs	msgSelection	MSEL
Subscription store	subStore	SS
Durable subscription state refresh interval	stateRefreshInt	SRI
Subscription cleanup level	cleanupLevel	CL
Subscription cleanup interval	cleanupInterval	CLINT
Subscription wildcard format	wildcardFormat	WCFMT
Publish acknowledgement window	pubAckInterval	PAI
Optimize for sparse subscription patterns	sparseSubs	SSUBS

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Broker queue manager	brokerQmgr	BQM
Client transport properties panel		
Certificate revocation list	sslCrl	CRL
Peer name	sslPeerName	SPEER
Reset count	sslResetCount	SRC
Receive exits	rcvExit	RCX
Receive exit initialization data	rcvExitInitData	RCXI
Send exits	sendExit	SDX
Send exit initialization data	sendExitInitData	SDXI
Security exit	secExit	SCX
Security exit initialization data	secExitInitData	SCXI
Other properties which are only on commands		
	localAddress	LA

WebSphere MQ Connection Factory property mappings:

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Basic panel		
Description	description	DESC
Queue manager	qmgrName	QMGR
Queue manager (when used with CCDT)	ccdtQmgrName	QMGR
Transport	wmqTransportType	TRAN
Hostname	qmgrHostname	HOST
Port	qmgrPortNumber	PORT
Server connection channel	qmgrSvrconnChannel	CHAN
Client ID	clientID	CID
Provider version	providerVersion	PVER
Client channel definition table URL	ccdtUrl	CCDT
Advanced properties panel		
Compress message headers	compressHeaders	HC
Compression algorithm for message payloads	compressPayload	MC
WebSphere MQ model queue name	modelQueue	TM
Temporary queue prefix	tempQueuePrefix	TQP
Temporary topic prefix	tempTopicPrefix	TTP
Retain messages, even if no matching consumer is available	msgRetention	MRET
Polling interval	pollingInterval	PINT
Rescan interval	rescanInterval	RINT
Maximum batch size	maxBatchSize	MBS

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Coded character set identifier	ccsid	CCS
Append an RFH version 2 header to reply messages	replyWithRFH2	TCM
Fail JMS method calls if the queue manager is quiescing	failIfQuiescing	FIQ
Broker properties panel		
Broker control queue	brokerCtrlQueue	BCON
Broker publication queue	brokerPubQueue	BPUB
Broker subscriber queue	brokerSubQueue	BSUB
Broker connection consumer subscription queue	brokerCCSubQueue	CCSUB
Version	brokerVersion	BVER
Specify where message selection occurs	msgSelection	MSEL
Subscription store	subStore	SS
Durable subscription state refresh interval	stateRefreshInt	SRI
Subscription cleanup level	cleanupLevel	CL
Subscription cleanup interval	cleanupInterval	CLINT
Subscription wildcard format	wildcardFormat	WCFMT
Publish acknowledgement window	pubAckInterval	PAI
Optimize for sparse subscription patterns	sparseSubs	SSUBS
Broker queue manager	brokerQmgr	BQM
Client transport properties panel		
Certificate revocation list	sslCrl	CRL
Peer name	sslPeerName	SPEER
Reset count	sslResetCount	SRC
Receive exits	rcvExit	RCX
Receive exit initialization data	rcvExitInitData	RCXI
Send exits	sendExit	SDX
Send exit initialization data	sendExitInitData	SDXI
Security exit	secExit	SCX
Security exit initialization data	secExitInitData	SCXI
Other properties which are only on commands		
	clonedSubs	CLS
	localAddress	LA

WebSphere MQ Queue property mappings:

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Basic panel		

Name on WebSphere Application Server administrative console panel	WebSphere Application Server command name	WebSphere MQ JMS Admin short name
Description	description	DESC
Queue name	queueName	QU
Queue manager or Queue sharing group name	qmgr	QMGR
Advanced properties panel		
Persistence	persistence	PER
Priority	priority	PRI
Expiry	expiry	EXP
Coded character set identifier	ccsid	CCS
Native encoding/Integer encoding/Decimal encoding/Floating point encoding	useNativeEncoding/integerEncoding/ decimalEncoding/floatingPointEncoding	ENC
Append RFH version 2 headers to messages sent to this destination	useRFH2	TC
Asynchronously send messages to the queue manager	sendAsync	PAALD
Read ahead, and cache, non-persistent messages for consumers	readAhead	RAALD
Read ahead consumer close method	readAheadClose	RACP

WebSphere MQ Topic property mappings

Name on WebSphere Application Server administrative console panel	WebSphereApplication Server command name	WebSphere MQ JMS Admin short name
Basic panel		
Description	description	DESC
Topic name	topicName	TOP
Queue manager or Queue sharing group name	qmgr	QMGR
Broker durable subscription queue	brokerDurSubQueue	BDSUB
Broker durable subscriber connection consumer queue	brokerCCDurSubQueue	CCDSUB
Broker publication queue	brokerPubQueue	BROKERPUBQ
Broker publication queue manager	brokerPubQmgr	BQM
Advanced properties panel		
Persistence	persistence	PER
Priority	priority	PRI
Expiry	expiry	EXP
Coded character set identifier	ccsid	CCS
Native encoding/Integer encoding/Decimal encoding/Floating point encoding	useNativeEncoding/integerEncoding/ decimalEncoding/floatingPointEncoding	ENC
Append RFH version 2 headers to messages sent to this destination	useRFH2	TC

Name on WebSphere Application Server administrative console panel	WebSphereApplication Server command name	WebSphere MQ JMS Admin short name
Asynchronously send messages to the queue manager	sendAsync	PAALD
Read ahead, and cache, non-persistent messages for consumers	readAhead	RAALD
Read ahead consumer close method	readAheadClose	RACP
Other properties which are only on commands		
	wildcardFormat	WCFMT
	brokerVersion	BVER

Chapter 9. Mail, URLs, and other J2EE resources

Using mail

You can enable your Java Platform, Enterprise Edition (Java EE) applications to use mail resources with the JavaMail API.

Before you begin

Using the JavaMail API, a code segment can be embedded in any Java EE application component, such as an EJB or a servlet, allowing the application to send a message and save a copy of the mail to the Sent folder.

The following is a code sample that you would embed in a Java EE application:

```
javax.naming.InitialContext ctx = new javax.naming.InitialContext();

   javax.mail.Session mail_session = (javax.mail.Session) ctx.lookup("java:comp/env/mail/MailSession3");
   MimeMessage msg = new MimeMessage(mail_session);

   msg.setRecipients(Message.RecipientType.TO, InternetAddress.parse("bob@coldmail.net"));

   msg.setFrom(new InternetAddress("alice@mail.eedge.com"));

   msg.setSubject("Important message from eEdge.com");

   msg.setText(msg_text);

   Transport.send(msg);


   Store store = mail_session.getStore();

   store.connect();

   Folder f = store.getFolder("Sent");

   if (!f.exists()) f.create(Folder.HOLDS_MESSAGES);

   f.appendMessages(new Message[] {msg});
```

About this task

Java EE applications can use JavaMail APIs by looking up references to logically named mail connection factories through the `java:comp/env/mail` subcontext that is declared in the application deployment descriptor and mapped to installation specific mail session resources. As in the case of other Java EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources.

1. Locate a resource through Java Naming and Directory Interface (JNDI). The Java EE specification considers a mail session instance as a resource, or a factory from which mail transport and store connections can be obtained. Do not hard code mail sessions (namely, fill up a Properties object, then use it to create a javax.mail.Session object). Instead, you must follow the Java EE programming model of configuring resources through the system facilities and then locating them through JNDI lookups.

   In the previous sample code, the line `javax.mail.Session mail_session = (javax.mail.Session) ctx.lookup("java:comp/env/mail/MailSession3");` is an example of not hard coding a mail session and using a resource name located through JNDI. You can consider the lookup name, `mail/MailSession3`, as an indirect reference to the real resource.

2. Define resource references while assembling your application. You must define a resource reference for the mail resource in the deployment descriptor of the component, because a mail session is referenced in the JNDI lookup. Typically, you can use an assembly tool that shipped with the application server.

   When you create this reference, be sure that the name of the reference matches the name used in the code. For example, the previous code uses `java:comp/env/mail/MailSession3` in its lookup. Therefore the name of this reference must be mail/Session3, and the type of the resource must be javax.mail.Session. After configuration, the deployment descriptor contains the following entry for the mail resource reference:

```
<resource-reference>
   <description>description</description>
   <res-ref-name>mail/MailSession3</res-ref-name>
   <res-type>javax.mail.Session</res-type>
   <res-auth>Container</res-auth>
</resource-reference>
```

3. Configure mail providers and sessions. The sample code references a mail resource, the deployment descriptor declares the reference, but the resource itself does not exist yet. Now you need to configure the mail resource that is referenced by your application component. Notice that the mail session you configure must have both its transport and mail access portions defined; the former required because the code is sending a message, the latter because it also saves a copy to the local mail store. When you configure the mail session, you need to specify a JNDI name. This is an important name for installing your application and linking up the resource references in your application with the real resources that you configure.
4. Install your application. You can install your application using either the administrative console or the scripting tool. During installation, the application server inspects all resource references and requires you to supply a JNDI name for each of them. This is not an arbitrary JNDI name, but the JNDI name given to a particular, configured resource that is the target of the reference.
5. Manage existing mail providers and sessions. You can update and remove mail providers and sessions.

   To update mail providers and sessions:

   a. Open the administrative console.
   b. Click **Resources** → **Mail** in the console navigation tree.
   c. Select the appropriate Java Mail resource to modify by clicking either **Mail Providers** or **Mail Sessions**.
   d. Select the specific resource to modify. To remove a mail provider or mail session, select the check box next to the appropriate resource and click **Delete**.
   e. Click **Apply** or **OK**.
   f. Save the configuration.
6. Optional: Debug a mail session.

What to do next

If your application has a client, you can update mail providers and mail sessions using the Application Client Resource Configuration Tool (ACRCT).

JavaMail API

The JavaMail APIs provide a framework that is platform and protocol independent for building mail client applications that are based on Java. The JavaMail APIs are generic for sending and reading mail. They require service providers, known in the application server as protocol providers, to interact with mail servers that run on pertaining protocols. For example, Simple Mail Transfer Protocol (SMTP) is a popular transport protocol for sending mail. Mail applications can connect to an SMTP server and send mail through it by using this SMTP protocol provider.

The application server supports the JavaMail API, Version 1.4. In the application server, the JavaMail API is supported in all Web application components, namely:

- servlets
- JavaServer Pages (JSP) files
- enterprise beans
- application clients

In addition to service providers, the JavaMail API requires the JavaBeans Application Framework (JAF) to handle mail content that is not plain text, including Multipurpose Internet Mail Extensions (MIME), URL pages, and file attachments.

The JavaMail APIs, the JAF, the service providers, and the protocols are shipped as part of the application server. The API and related specifications are repackaged from materials that are licensed to Sun Microsystems.

Note: If you are using Java 5 to run your code, you will need the JAF 1.1 package. For Java 6 and later, the JAF package is part of the run time environment.

Mail providers and mail sessions

A mail service provider is a driver that supports mail interaction with mail servers that use a particular mail protocol. The application server includes service providers, which are also known as protocol providers, for mail protocols.

A mail provider encapsulates a collection of protocol providers. For example, the application server has a built-in mail provider that encompasses the most common protocol providers. These protocol providers are installed as the default and suffice for most applications. If you have a particular application that requires custom protocol providers, follow the steps that are outlined in the chapter on mail sessions in the JavaMail API Design Specification to install your own protocol providers.

Mail sessions are represented by the javax.mail.Session class. A mail session object authenticates users and controls access to messaging systems.

To create mail applications that are platform independent, use a resource factory reference to create a mail session. A resource factory is an object that provides access to resources in the deployed environment of a program. Resource factories use the naming conventions that are defined by the Java Naming and Directory Interface (JNDI).

Note: Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.

JavaMail security permissions best practices

In many of its activities, the JavaMail API needs to access certain configuration files. The JavaMail and JavaBeans Activation Framework binary packages themselves already contain the necessary configuration files. However, the JavaMail API allows the user to define user-specific and installation-specific configuration files to meet special requirements.

The two locations where you can place these configuration files are the <user.home> and <java.home>/lib directories. For example, if the JavaMail API needs to access a file named mailcap when it sends a message, the API:

1. Tries to access <user.home>/mailcap.
2. If the first attempt fails due to a lack of security permission or a nonexistent file, the API searches in <java.home>/lib/mailcap.

3. If the second attempt also fails, the API searches in the META-INF/mailcap location in the class path. This location actually leads to the configuration files contained in the mail-impl.jar and activation-impl.jar files.

Application Server uses JavaMail API configuration files that are contained in the mail-impl.jar and activation-impl.jar files, and there are no mail configuration files in <user.home> and <java.home>/lib directories. To ensure proper functioning of the JavaMail API, Application Server grants *file read* permission for both the mail-impl.jar and activation-impl.jar files to all of the installed applications.

JavaMail code attempts to access configuration files at <user.home> and <java.home>/lib, which can cause an access control exception to be thrown, since the default configuration does not grant file read permission for those two locations by default. This activity does not affect the proper functioning of the JavaMail API, but you might see a large amount of mail-related security exceptions reported in the system log, and these errors could overshadow harmful errors for which you are looking. This is a sample of the security message, SECJ0314W:

```
[02/31/08 12:55:38:188 PDT] 00000058 SecurityManag W   SECJ0314W: Current Java 2 Security policy reported a
potential violation of Java 2 Security Permission.
Please refer to Problem Determination Guide for further information.

Permission:

      D:\o063919\java\jre\lib\javamail.providers : access denied (java.io.FilePermission
D:\o063919\java\jre\lib\javamail.providers read)


Code:

     com.ibm.ws.mail.SessionFactory  in  {file:/D:/o063919/lib/runtime.jar}



Stack Trace:

java.security.AccessControlException: access denied (java.io.FilePermission D:\o063919\java\jre\lib\javamail.providers read)
 at java.security.AccessControlContext.checkPermission(AccessControlContext.java(Compiled Code))
 at java.security.AccessController.checkPermission(AccessController.java(Compiled Code))
 at java.lang.SecurityManager.checkPermission(SecurityManager.java(Compiled Code))
 at com.ibm.ws.security.core.SecurityManager.checkPermission(SecurityManager.java(Compiled Code))
 at java.lang.SecurityManager.checkRead(SecurityManager.java(Compiled Code))
 at java.io.FileInputStream.<init>(FileInputStream.java(Compiled Code))
 at java.io.FileInputStream.<init>(FileInputStream.java:89)
 at javax.mail.Session.loadFile(Session.java:1004)
 at javax.mail.Session.loadProviders(Session.java:861)
 at javax.mail.Session.<init>(Session.java:191)
 at javax.mail.Session.getInstance(Session.java:213)
 at com.ibm.ws.mail.SessionFactory.getObjectInstance(SessionFactory.java:67)
 at javax.naming.spi.NamingManager.getObjectInstance(NamingManager.java:314)
 at com.ibm.ws.naming.util.Helpers.processSerializedObjectForLookupExt(Helpers.java:894)
 at com.ibm.ws.naming.util.Helpers.processSerializedObjectForLookup(Helpers.java:701)
 at com.ibm.ws.naming.jndicos.CNContextImpl.processResolveResults(CNContextImpl.java:1937)
 at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1792)
 at com.ibm.ws.naming.jndicos.CNContextImpl.doLookup(CNContextImpl.java:1707)
 at com.ibm.ws.naming.jndicos.CNContextImpl.lookupExt(CNContextImpl.java:1412)
 at com.ibm.ws.naming.jndicos.CNContextImpl.lookup(CNContextImpl.java:1290)
 at com.ibm.ws.naming.util.WsnInitCtx.lookup(WsnInitCtx.java:145)
 at javax.naming.InitialContext.lookup(InitialContext.java:361)
 at emailservice.com.onlinebank.bpel.EmailService20060907T224337EntityAbstractBase$JSE_6.
execute(EmailService20060907T224337EntityAbstractBase.java:32)
 at com.ibm.bpe.framework.ProcessBase6.executeJavaSnippet(ProcessBase6.java:256)
 at emailservice.com.onlinebank.bpel.EmailService20060907T224337EntityBase.invokeSnippet
(EmailService20060907T224337EntityBase.java:40)
```

Note: If this situation is a problem, consider adding more read access permissions for more locations. This should eliminate most, if not all, JavaMail-related harmless security exceptions from the log file.

The permissions required by JavaMail are as follows:

```
grant codeBase "file:${application}" {
  // Allow access to default configuration files
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}javamail.address.map", "read";
```

```
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}javamail.providers", "read";
  permission java.io.FilePermission "${java.home}${/}jre${/}lib${/}mailcap", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}javamail.address.map", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}javamail.providers", "read";
  permission java.io.FilePermission "${java.home}${/}lib${/}mailcap", "read";
  permission java.io.FilePermission "${user.home}${/}.mailcap", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}activation-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}lib${/}mail-impl.jar", "read";
  permission java.io.FilePermission "${was.install.root}${/}plugins${/}com.ibm.ws.prereq.javamail.jar", "read";
  // If using an isolated mail provider,
  // add additional file read permissions for each jar defined
  // for the isolated mail provider
  // permission java.io.FilePermission "path${/}mail.jar, "read";

  // Allow connection to mail server using SMTP
  permission java.net.SocketPermission "*:25", "connect,resolve";
  // Allow connection to mail server using SMTPS
  permission java.net.SocketPermission "*:465", "connect,resolve";

  // Allow connection to mail server using IMAP
  permission java.net.SocketPermission "*:143", "connect,resolve";
  // Allow connection to mail server using IMAPS
  permission java.net.SocketPermission "*:993", "connect,resolve";

  // Allow connection to mail server using POP3
  permission java.net.SocketPermission "*:110", "connect,resolve";
  // Allow connection to mail server using POP3S
  permission java.net.SocketPermission "*:995", "connect,resolve";

  // Allow System.getProperties() to be used
  // permission java.util.PropertyPermission "*", "read,write";
  // Otherwise use the following to allow system properties to be read
  permission java.util.PropertyPermission "*", "read";
};
```

Mail: Resources for learning

Use the following links to find relevant supplemental information about the JavaMail API. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming model and decisions

- JavaMail documentation

Programming specifications

- JavaMail 1.3 API documentation (Sun Java specifications)

JavaMail support for IPv6

WebSphere Application Server and its JavaMail component support Internet Protocol Version 6.0 (IPv6), meaning that:

- Both can run on a pure IPv4 network, a pure IPv6 network, *or* a mixed IPv4 and IPv6 network.
- On either the pure IPv6 network or the mixed network, the JavaMail component works with mail servers (such as the SMTP mail transfer agent, and the IMAP and POP3 mail stores) that are also IPv6 compatible. Additionally, a JavaMail component that is run on the mixed IPv4 and IPv6 network can communicate with mail servers using IPv4.

Use of brackets with IPv6 addresses

When you configure a mail session, you can specify the mail server hosts (also known as mail transport and mail store hosts) with domain-qualified host names or numerical IP addresses. Using host names is generally the preferred method. If you use IP addresses, however, consider enclosing IPv6 addresses in square brackets to prevent parsing inaccuracies. See the following example:

```
[fe80::202:57ff:fec4:2334]
```

The JavaMail API requires a combination of many host names or IP addresses with a port number, using the `host:port number` syntax . This extra colon can cause the port number to be read as part of an IPv6 address. Using brackets prevents your JavaMail implementation from processing the extra characters erroneously.

Using URL resources within an application

Java Platform, Enterprise Edition (Java EE) applications can use Uniform Resource Locators (URLs) by looking up references to logically named URL connection factories through the `java:comp/env/url` subcontext, which is declared in the application deployment descriptor and mapped to installation specific URL resources.

About this task

As in the case of other Java EE resources, this can be done in order to eliminate the need for the application to hard code references to external resources. The process is the same used with other Java EE resources, such as JDBC objects and JavaMail sessions.

1. Develop an application that relies on naming features.
2. Define resource references while assembling your application. A URL resource that uses a built-in protocol, such as HTTP, FTP, or file, can use the default URL provider. URL resources that use other protocols need to use a custom URL provider.
3. Configure your URL resources within an application.
   a. Open the administrative console.
   b. Click **Resources**>**URL** in the console navigation tree.
   c. Click either **URL Providers** or **URLs** to modify the appropriate resource.
4. Optional: Configure URL providers and URLs within an application client using the Application Client Resource Configuration Tool (ACRCT).
5. Manage URL providers and URL resources used by the deployed application. To update or remove existing URL configurations:
   a. Open the administrative console.
   b. Click **Resources** > **URL** in the console navigation tree.
   c. Click either **URL Providers** or **URLs** to modify the appropriate resource.
   d. Select the URL to modify.
   e. Modify the URL properties.
   f. Click **Apply** or **OK**.

   To remove URL providers and URLs, after step 2, click *URL_provider* > **URLs**. Select the URL you want to remove and click **Delete**. Then, click **Apply** or **OK**.

URLs

A Uniform Resource Locator (URL) is an identifier that points to an electronically accessible resource, such as a directory file on a machine in a network, or a document stored in a database.

URLs appear in the format *scheme*:*scheme_information*.

You can represent a *scheme* as HTTP, FTP, file, or another term that identifies the type of resource and the mechanism by which you can access the resource.

In a World Wide Web browser location or address box, a URL for a file available using HyperText Transfer Protocol (HTTP) starts with `http:`. An example is `http://www.ibm.com`. Files available using File Transfer Protocol (FTP) start with `ftp:`. Files available locally start with `file:`.

The *scheme_information* commonly identifies the Internet machine making a resource available, the path to that resource, and the resource name. The *scheme_information* for HTTP, FTP and file generally starts with two slashes (//), then provides the Internet address separated from the resource path name with one slash (/). For example,

`http://www.ibm.com/software/webservers/appserv/library.html`.

For HTTP and FTP, the path name ends in a slash when the URL points to a directory. In such cases, the server generally returns the default index for the directory.

URL provider collection

Use this page to view existing URL providers, which supply the implementation classes that are necessary for WebSphere Application Server to access a URL through a specific protocol. The default URL provider provides connectivity through protocols that are supported by the IBM Developer Kit for the Java™ Platform, compatible with the Java 2 Standard Edition Platform 1.3.1. These protocols include HyperText Transfer Protocol (HTTP) and File Transfer Protocol (FTP), which work for must URLs.

To view this administrative console page, click **Resources** > **URL** > **URL Providers**.

Name

Specifies the administrative name for the URL provider.

Scope

Specifies the scope of this URL provider, which can support multiple URL configurations. All of the URL configurations that are supported by this provider inherit this scope.

Description

Describes the URL provider for your administrative records.

URL provider settings

Use this page to configure URL providers, which support WebSphere Application Server connections to a URL over a specific protocol.

To view this administrative console page, click **Resources** > **URL** > **URL Providers** > *URL_provider*.

Scope

Specifies the scope of this URL provider, which can support multiple URL configurations. All of the URL configurations that are supported by this provider inherit this scope.

Name

Specifies the administrative name for the URL provider.

Description

Describes the URL provider, for your administrative records.

Class path

Specifies paths or JAR file names which together form the location for the resource provider classes.

Stream handler class name
Specifies fully qualified name of a user-defined Java class that extends the `java.net.URLStreamHandler` class for a particular URL protocol, such as FTP.

Protocol
Specifies the protocol supported by this stream handler. For example, NNTP, SMTP, FTP.

URL collection
Use this page to view existing Uniform Resource Locator (URL) configurations, which are sets of properties that define WebSphere Application Server connections to URLs. URLs are location names that represent electronically accessible resources, such as a directory file on a machine in a network or a document stored in a database.

You can access this administrative console page in one of two ways:

- **Resources** > **URL Providers** > *URL_provider* > **URLs**
- **Resources** > **URL** > **URLs**

Name
Specifies the display name for the resource.

JNDI Name
Specifies the JNDI name.

Scope
Specifies the scope of the URL provider that supports this URL configuration. Only applications that are installed within this scope can use this URL configuration to access URL resources.

Provider
Specifies the URL provider that supplies the implementation classes for using a specific protocol to access this URL.

Description
Specifies the description of the resource.

Category
Specifies the category string, which you can use to classify or group the resource.

URL configuration settings
Use this page to define connections to Uniform Resource Locators (URLs), which are location names that represent electronically accessible resources. A collection of URL connection properties is often called a URL configuration in the WebSphere Application Server environment. The targeted resources are remote to your Application Server installation.

You can access this administrative console page in one of two ways:

- **Resources** > **URL** > **URLs** > *URL*
- **Resources** > **URL** > **URL Providers** > *URL_provider* > **URLs** > *URL*

Scope
Specifies the scope of the URL provider that supports this URL configuration. Only applications that are installed within this scope can use this URL configuration to access URL resources.

Provider
Specifies the URL provider that WebSphere Application Server uses for this URL configuration.

Note: If you previously defined one or more URL providers at the relevant scope, you see a list from which you can select an existing URL provider for your new URL configuration.

Create New Provider
Provides the option of configuring a new URL provider for the new URL configuration.

Create New Provider is displayed only when you create a new URL from the **Resources** > **URL** > **URLs** path. In this flow, you can create a new URL provider if needed. The URL provider can not be changed during an edit.

Clicking **Create New Provider** triggers the console to display the URL provider configuration page, where you create a new provider. After you click **OK** to save your settings, you see the URL collection page. Click **New** to define a new URL configuration for use with the new provider; the console now displays a configuration page that lists the new provider as the URL configuration Provider.

Name
Specifies the display name for the resource.

JNDI Name
Specifies the JNDI name.

Note: Adhere to the following requirements for JNDI names:

- Do not assign duplicate JNDI names across different resource types (such as mail sessions versus URL configurations).
- Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.

Description
Specifies the description of the resource.

Category
Specifies the category string, which you can use to classify or group the resource.

Specification
Specifies the string from which to form a URL.

URLs: Resources for learning
Use the following links to find relevant supplemental information about URLs. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming specifications
- W3C Architecture - Naming and Addressing: URIs, URLs
- URL API documentation

Mapping logical names of environment resources to their physical names

This topic provides instructions on configuring *new* resource environment entries, which define environment resources that are the binding targets for resource-environment-references in an application's deployment descriptor.

1. Configure a resource environment provider, which is a library that provides the implementation for an environment resource factory. In the administration console, begin by clicking **Resources > Resource Environment > Resource Environment Providers > New**. (See the New Resource Environment Provider topic for more information.)
2. After saving your resource environment provider, go to the Additional Properties heading and click **Resource environment entries**. Click **New** to define a new resource environment entry. Refer to the "Resource environment entry settings" on page 2018 topic for descriptions of the required fields.
3. You also might need to create a referenceable, which specifies the factory class name that converts information in the name space into a class instance for your resource. To view the appropriate administrative console page for referenceables, click **Resources > Resource Environment > Resource Environment Providers >** *your_resource_environment_provider* > **Referenceables**. Click **New** to begin the configuration process. See the "Referenceables settings" on page 2020 topic for descriptions of the required fields.

Resource environment providers and resource environment entries

A resource environment reference maps a logical name used by the client application to the physical name of an object.

Not all objects bound into the server JNDI namespace are intended for use by an application client. For example, the WebSphere Application Server client run time does not support the use of Java 2 Connector (J2C) objects on the client. The object needs to be remotable, and the client-side implementations must be made available on the application client run-time classpath.

Resource environment references are different than resource references. Resource environment references allow your application client to use a logical name to look up a resource bound into the server JNDI namespace. A resource reference allows your application to use a logical name to look up a local J2EE resource. The J2EE specification does not specify a particular implementation of a resource.

Resource environment provider collection

Use this page to view resource environment providers, which encapsulate the referenceables that convert resource environment entry data into resource objects.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers**.

Name

Specifies a text identifier for the resource environment provider.

Data type String

Scope

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Description

Specifies a text string describing the resource environment provider.

Data type String

Resource environment provider settings

Use this page to create settings for a resource environment provider.

To view this administrative console page, click **Resources** > **Resource environment** > **Resource environment providers** > *resource environment provider*.

Scope:

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Name:

Specifies the name of the resource provider.

Data type String

Description:

Specifies a text description for the resource provider.

Data type String

New Resource environment provider

Use this page to define the configuration for a library that provides the implementation for a environment resource factory.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers > New**.

Scope:

Specifies the scope of this resource environment provider, which automatically becomes the scope of the resource environment entries that you define with this provider.

Name:

Specifies a text identifier for the resource environment provider.

Data type String

Description:

Specifies a text string describing the resource environment provider.

Data type String

Resource environment entries collection

Use this page to view configured resource environment entries. Within an application server name space, the data contained in a resource environment entry is converted into an object that represents a physical resource. This resource is frequently called an *environment resource*.

An environment resource can be of any arbitrary type. See the latest EJB specification for more information about resource environment references and environment resources.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Environment** > **Resource environment entries**
- **Resources** >**Resource Environment** > **Resource Environment Providers** > *resource_environment_provider* > **Resource Environment Entries**

Name
Specifies a text identifier that helps distinguish this resource environment entry from others.

For example, you can use *My Resource* for the name.

Data type String

JNDI Name
Specifies the string to be used when looking up this environment resource using JNDI.

This is the string to which you bind resource environment reference deployment descriptors.

Data type String

Scope
Specifies the resource environment entry scope, which is inherited from the resource environment provider.

Provider
Specifies the resource environment provider for this entry. The provider encapsulates the classes that, when implemented, convert resource environment entry data into resource objects.

Description
Specifies text for information to help further identify and distinguish this resource

Data type String

Category
Specifies a category you can use to group environment resources according to some common feature.

It is strictly an organizational property and has no effect on the function of the environment resource.

Data type String

Resource environment entry settings
Use this page to configure resource environment entries. Within an application server name space, the data contained in a resource environment entry is converted into an object that represents a physical resource. Rather than represent a connection factory, which provides connections to a resource, this object *directly* represents a resource. This design can make the resource available to application modules that do not run entirely on the application server. Examples include some application clients and Web modules.

You can access this administrative console page in one of two ways:

- **Resources** > **Resource Environment** > **Resource environment entries** > *resource_environment_entry*
- **Resources** >**Resource Environment** > **Resource Environment Providers** > *resource_environment_provider* > **Resource Environment Entries** > *resource_environment_entry*

Scope:

Specifies the scope of the resource environment provider, which is a library that supplies the implementation class for a resource environment factory. Within a JNDI name space, WebSphere Application Server uses the factory to transform your resource environment entry into an object that directly represents a physical resource.

Provider:

Specifies the resource environment provider.

Provider shows all of the existing resource environment providers that are defined at the relevant scope. Select one from the list if you want to use an existing resource environment provider as Provider.

Name:

Specifies a display name for the resource.

Data type String

JNDI name:

Specifies the JNDI name for the resource, including any naming subcontexts.

This name is used as the linkage between the platform's binding information for resources defined by a module's deployment descriptor and actual resources bound into JNDI by the platform.

Data type String

Note: Adhere to the following requirements for JNDI names:

- Do not assign duplicate JNDI names across different resource types (such as resource environment entries versus J2C connection factories).
- Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.

Description:

Specifies a text description for the resource.

Data type String

Category:

Specifies a category string that you can use to classify or group the resource.

Data type String

Referenceables:

Specifies the referenceable, which encapsulates the class name of the factory that converts resource environment entry data into a class instance for a physical resource.

Data type Drop-down menu

Referenceables collection

Use this page to view configured referenceables, which encapsulate the class name of the factory that converts information in the name space into a class instance for a physical resource.

To view this administrative console page, click **Resources** > **Resource environment** > **Resource Environment Providers** > *resource_environment_provider* > **Referenceables**.

Factory Class name
Specifies a javax.naming.spi.ObjectFactory implementation name

Data type String

Class name
Specifies the package name of the referenceable, for example: `javax.naming.Referenceable`

Data type String

Referenceables settings
Use this page to set the class name of the factory that converts information in the name space into a class instance of a physical resource.

To view this administrative console page, click **Resources > Resource Environment > Resource Environment Providers >** *resource_environment_provider* **> Referenceables >** *referenceable*.

Factory class name:

Specifies a javax.naming.ObjectFactory implementation class name

Data type String

Class name:

Specifies the Java type to which a Referenceable provides access, for binding validation and to create the reference.

Data type String

Resource environment references
Use this page to designate how the resource environment references of application modules map to remote resources, which are represented in the product as resource environment entries.

To view this administrative console page, click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **Resource environment references**.

Each row of the table depicts a resource environment reference within a specific module of your application. If you bound any references to resource environment entries during application assembly, you see the JNDI names of those resource environment entries in the applicable rows.

To set the mapping relationships between your resource environment references and resource environment entries:

1. Select a row. Be aware that if you check multiple rows on this page, the resource mapping target that you select in step 2 applies to all of those references.
2. Click **Browse** to select a resource environment entry from the new page that is displayed, the Available Resources page. The Available Resources page shows all resource environment entries that are available mapping targets for your application references.
3. Click **Apply**. The console displays the Resource environment references page again. In the rows that you previously selected, you now see the JNDI name of the new resource mapping target.
4. Repeat the previous steps as necessary.
5. Click **OK**. You now return to the general configuration page for your enterprise application.

Table column heading descriptions:

Select
Select the check boxes of the rows that you want to edit.

Module
The name of a module in the application.

EJB
The name of an enterprise bean that is accessed by the module.

URI
Specifies location of the module relative to the root of the application EAR file.

Reference binding
The name of a resource environment reference that is declared in the deployment descriptor of the application module. The reference corresponds to a resource that is bound as a resource environment entry into the JNDI name space of the application server.

JNDI name
The Java Naming and Directory Interface (JNDI) name of the resource environment entry that is the mapping target of the resource environment reference.

Data type String

Configuring mail providers and sessions

Configure your own mail providers and sessions to customize how mail is handled in the application server. A mail provider encapsulates a collection of protocol providers, like SMTP, IMAP and POP3, and others. Mail sessions authenticate users and control access to messaging systems.

About this task

The application server includes a default mail provider that is called the built-in provider. If you use the default mail provider, you only have to configure the mail session.

To use a customized mail provider, you must create the mail session and provider.

- Create the mail session.
  1. In the administrative console, click **Resources** → **Mail** → **Mail sessions**.
  2. Select the scope for the new mail session.
  3. Click **New**.
  4. Type the mail session name in the Name field.
  5. Type the JNDI name in the JNDI Name field.
  6. Optional: Enable strict Internet address parsing. This option specifies whether the recipient addresses must be parsed in strict compliance with RFC 822, which is a specifications document that is issued by the Internet Architecture Board. This setting is not generally used for most mail applications, and by default this setting is not enabled.

     RFC 822 syntax for parsing addresses effectively enforces a strict definition of a valid e-mail address. If you select this setting, your mail component adheres to RFC 822 syntax and rejects recipient addresses that do not parse into valid e-mail addresses as defined by the specification. If you do not select this setting, your mail component does not adhere to RFC 822 syntax and accepts recipient addresses that do not comply with the specification. You can view the RFC 822 specification at the web site for the World Wide Web Consortium.

7. Optional: Enable debug mode. Select this option to print interaction between the mail application and the mail servers and the properties of this mail session to the SystemOut.log file.
8. Provide information for the incoming mail service, outgoing service, or both. Enter the following information in the fields that are provided:
   - Server
   - Protocol
   - User
   - Password
   - Return e-mail address. This field is available for outgoing mail properties.
9. Click **Apply** or **OK**.

- Create the mail provider, and optionally define one or more protocol providers. In the administrative console, click **Resources** → **Mail** → **Mail Providers**.
  1. Select the scope for the new mail provider.
  2. Click **New**.
  3. Type the name of the mail provider in the name field.
  4. Optional: Isolate the mail provider.

     Note: You can isolate a mail provider to allow different versions of the same provider to be loaded in the same Java Virtual Machine (JVM). For example, you might want to deploy multiple applications on a single server, but each application requires different versions or implementations of the mail provider. You can now isolate each version or implementation of the provider, and the provider will be loaded in its own class loader and will not interfere with other implementations. There are some general considerations for isolating any type of resource provider; refer to the topic on considerations for isolated resource providers for more information.

     a. Select **Isolate this mail provider**.
     b. Give the mail provider a unique class path that is appropriate for that version or implementation.
  5. Click **Apply** or **OK**.
  6. Optional: Define one or more protocol providers for the mail provider.
     a. Click *mail_provider*.
     b. Click **Protocol Providers**.
     c. Click **New**.
     d. Type the protocol name in the **Protocol** field.
     e. Type the class name in the **Class name** field.
     f. Select the type of mail server that this protocol provider supports. Select TRANSPORT or STORE. TRANSPORT corresponds to outgoing mail services, and STORE corresponds to incoming mail services.
     g. Click **Apply** or **OK**.

     Ensure that every mail session is defined under a parent mail provider. Select a mail provider first and then create your new mail session.
- Optional: Configure the mail session.
  1. Click *mail_provider*.
  2. Click **Mail Sessions**.
  3. Click *mail_session*.
  4. Make changes to appropriate fields.
  5. Click **Apply** or **OK**.

What to do next

If your application has a client, you can configure mail providers and sessions using the Application Client Resource Configuration Tool.

Mail provider collection

Use this page to view available JavaMail service providers, also known as *mail providers*. The mail provider encapsulates a collection of protocol providers, which implement the protocols for communication between your mail application and mail servers.

To view this administrative console page, click **Resources** > **Mail** > **Mail Providers**.

The **built-in mail provider** made available by WebSphere Application Server encompasses three protocol providers: Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP) and Post Office Protocol (POP3). Select the built-in provider if these protocols provide the right support for your mail system. If you have installed or plan to install different protocol providers, you must assign them a mail provider; select the scope at which you want the new mail provider to implement the protocols, then select **New**.

Name
Specifies the name of the JavaMail resource provider.

Scope
Specifies the scope in which the mail provider supports installed mail applications.

Description
Specifies the resource provider description.

Mail provider settings

Use this page to edit mail provider properties or configure a new mail provider. The mail provider encapsulates a collection of protocol providers, which implement the protocols for communication between your mail application and mail servers.

To view this administrative console page, click **Resources** → **Mail** → **Mail Providers** → ***mail_provider***, or create a new mail provider by clicking **Resources** → **Mail** → **Mail Providers** → **New**.

Scope
Specifies the scope in which the mail provider supports installed mail applications.

Name
Specifies the name of the mail provider.

Description
Specifies the resource provider description.

Isolate this mail provider
Specifies that this mail provider will be loaded in its own class loader. This allows the application server to load different versions or implementations of the same mail provider in the same Java Virtual Machine. Give each version or implementation of the mail provider a unique class path that is appropriate for that version or implementation.

Class path
Specifies the class path to a Java archive (JAR) file that contains the implementation classes for this mail provider. If more than one JAR file provides the complete implementation, add an entry for each JAR file that the mail provider requires. Enter one class path per line; do not use class path separator information.

Protocol providers collection

Use this page to select or add a protocol provider that supports interaction between your mail application and mail servers. For example, your application might require the Simple Mail Transfer Protocol (SMTP), which is a popular transport protocol for sending mail. Selecting that protocol provider allows your mail application to connect and send mail through the server.

To view this administrative console page, click **Resources** → **Mail** → **Mail Providers** → ***mail_provider*** → **Protocol Providers**.

Protocol

Specifies the configuration of the protocol provider for a given protocol.

Class name

Specifies the implementation class for the specific protocol provider, which is also known as the mail service provider.

Type

Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Protocol providers settings

Use this page to set properties of a protocol provider, which provides the implementation class for a specific protocol to support communication between your mail application and mail servers.

Note: The application server contains protocol providers for various types of protocols, including SMTP, IMAP and POP3. If you require custom providers for protocols that are not provided by the application server, install them in your application serving environment before configuring the providers. See the JavaMail API design specification for guidelines. After configuring your protocol providers, return to the mail provider page to find the link for configuring mail sessions.

To view this administrative console page, click **Resources** → **Mail** → **Mail Providers** → ***mail_provider*** → **Protocol Providers** → ***protocol_provider***.

Scope

Specifies the scope at which this protocol provider was created. Only applications that are installed within this scope can use a protocol that is configured according to the settings of this protocol provider.

Protocol

Specifies the configuration of the protocol provider for a given protocol.

Class name

Specifies the implementation class of this protocol provider.

Type

Specifies the type of protocol provider. Valid options are **STORE** or **TRANSPORT**.

Mail session collection

Use this page to view mail sessions that are defined under the parent mail provider.

You can access this administrative console page in one of two ways:

- **Resources** > **Mail** > **Mail Sessions**
- **Resources** > **Mail** > **Mail Providers** > *mail_provider* > **Mail Sessions**

Name

Specifies the administrative name of the JavaMail session object.

Scope
Specifies the scope at which the mail session was created. Only JavaMail applications that are installed in this scope can use this mail session.

Provider
Specifies the mail provider that WebSphere Application Server uses for this mail session.

JNDI Name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Description
Specifies an optional description for your administrative records.

Category
Specifies an optional collection for classifying or grouping sessions.

Mail session settings
Use this page to configure mail sessions.

You can access this administrative console page in one of two ways:
- **Resources** → **Mail** → **Mail sessions** → ***mail_session***
- **Resources** → **Mail** → **Mail Providers** → ***mail_provider*** → **Mail sessions** → ***mail_session***

Scope
Specifies the scope of the mail provider that implements the JavaMail API for this mail session. Only applications that you installed within this scope can use this mail session.

Provider
Specifies the mail provider that the application server uses for this mail session.

When you create a mail session, if you previously defined one or more mail providers at the relevant scope, you will see a list from which you can select an existing mail provider for the new mail session.

Create New Provider
Provides the option of configuring a new mail provider for the new mail session.

Create New Provider is displayed only when you click **Resources** → **Mail** → **Mail sessions** → **New** to create a new mail session.

Clicking **Create New Provider** triggers the console to display the mail provider configuration page, where you create a new provider. After you click **OK** to save your settings, you see the mail session collection page. Click **New** to define a new mail session for use with the new provider; the console now displays a configuration page that lists the new mail provider as the mail session Provider.

Note: After you create a mail session, you cannot change the provider of that mail session.

Name
Specifies the administrative name of the JavaMail session object.

JNDI name
Specifies the Java Naming and Directory Interface (JNDI) name for the resource, including any naming subcontexts.

This name provides the link between the platform binding information for resources that are defined in the client application deployment descriptor and the actual resources bound into JNDI by the platform.

Note: Adhere to the following requirements for JNDI names:

- Do not assign duplicate JNDI names across different resource types (such as mail sessions versus data sources).
- Do not assign duplicate JNDI names for multiple resources of the same type in the same scope.

Description

Specifies an optional description for your administrative records.

Category

Specifies an optional collection for classifying or grouping sessions.

Enable debug mode

Toggles debug mode on and off for this mail session.

Enable strict Internet address parsing

Specifies whether the recipient addresses must be parsed in strict compliance with RFC 822, which is a specifications document issued by the Internet Architecture Board.

This setting is not generally used for most mail applications. RFC 822 syntax for parsing addresses effectively enforces a strict definition of a valid e-mail address. If you select this setting, your mail component adheres to RFC 822 syntax and rejects recipient addresses that do not parse into valid e-mail addresses (as defined by the specification). If you do not select this setting, the mail component does not adhere to RFC 822 syntax and accepts recipient addresses that do not comply with the specification. By default, this setting is not selected. You can view the RFC 822 specification at the World Wide Web Consortium web site.

Outgoing Mail Properties

Server:

Specifies the server that is accessed when sending mail.

Protocol:

Specifies the protocol to use when sending mail. Actual protocol values are defined in the protocol providers that you configured for the current mail provider.

User:

Specifies the user of the mail account when the outgoing mail server requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Password:

Specifies the password to use when the outgoing mail server requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Verify Password:

Verify the password.

Return e-mail address:

Specifies the Internet e-mail address that is displayed in messages as the mail originator.

This value represents the Internet e-mail address that, by default, displays in the received message in the **From** or the **Reply-To** address. The recipient's reply will come to this address.

Incoming Mail Properties

Server:

Specifies the server that is accessed when receiving mail.

Protocol:

Specifies the protocol to use when receiving mail. Actual protocol values are defined in the protocol providers that you configured for the current mail provider.

User:

Specifies the user of the mail account when the incoming mail server requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Password:

Specifies the password to use when the incoming mail server requires authentication.

This setting is not generally used for most mail servers. Leave this field blank unless you use a mail server that requires a user ID and password.

Verify Password:

Verify the password.

Configuring mail, URLs, and resource environment entries with scripting

Use scripting to configure mail, URLs, and resource environment entries.

About this task

This topic contains the following tasks:

Configuring new mail providers using scripting

You can use scripting and the wsadmin tool to configure new mail providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new mail provider:

1. Identify the parent ID:
   - Using Jacl:
     ```
     set node [$AdminConfig getid /Cell:mycell/Node:mynode/]
     ```
   - Using Jython:
     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:
   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```
2. Get required attributes:
   - Using Jacl:
     ```
     $AdminConfig required MailProvider
     ```
   - Using Jython:
     ```
     print AdminConfig.required('MailProvider')
     ```

   Example output:
   ```
   Attribute       Type
   name       String
   ```
3. Set up required attributes:
   - Using Jacl:
     ```
     set name [list name MP1]
     set mpAttrs [list $name]
     ```
   - Using Jython:
     ```
     name = ['name', 'MP1']
     mpAttrs = [name]
     ```
4. Create the mail provider:
   - Using Jacl:
     ```
     set newmp [$AdminConfig create MailProvider $node $mpAttrs]
     ```
   - Using Jython:
     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:
   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new mail sessions using scripting

You can use scripting and the wsadmin tool to configure new mail sessions.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new mail session:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```
   - Using Jython:

     ```
     newmp = AdminConfig.getid('/Cell:mycell/Node:mynode/MailProvider:MP1/')
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```
2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required MailSession
     ```
   - Using Jython:

     ```
     print AdminConfig.required('MailSession')
     ```

   Example output:

   ```
   Attribute       Type
   name       String
   jndiName       String
   ```
3. Set up required attributes:
   - Using Jacl:

     ```
     set name [list name MS1]
     set jndi [list jndiName mail/MS1]
     set msAttrs [list $name $jndi]
     ```

     Example output:

     ```
     {name MS1} {jndiName mail/MS1}
     ```
   - Using Jython:

     ```
     name = ['name', 'MS1']
     jndi = ['jndiName', 'mail/MS1']
     msAttrs = [name, jndi]
     print msAttrs
     ```

     Example output:

     ```
     [[name, MS1], [jndiName, mail/MS1]]
     ```
4. Create the mail session:
   - Using Jacl:

     ```
     $AdminConfig create MailSession $newmp $msAttrs
     ```
   - Using Jython:

     ```
     print AdminConfig.create('MailSession', newmp, msAttrs)
     ```

   Example output:

```
MS1(cells/mycell/nodes/mynode|resources.xml#MailSession_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new protocols using scripting

You can configure new protocols with scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new protocol:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

     Example output:

     ```
     MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
     ```

2. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ProtocolProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ProtocolProvider')
     ```

     Example output:

     ```
     Attribute       Type
     protocol      String
     classname       String
     ```

3. Set up required attributes:
   - Using Jacl:

     ```
     set protocol [list protocol "Put the protocol here"]
     set classname [list classname "Put the class name here"]
     set ppAttrs [list $protocol $classname]
     ```

     Example output:

     ```
     {protocol protocol1} {classname classname1}
     ```

   - Using Jython:

     ```
     protocol = ['protocol', "Put the protocol here"]
     classname = ['classname', "Put the class name here"]
     ppAttrs = [protocol, classname]
     print ppAttrs
     ```

     Example output:

     ```
     [[protocol, protocol1], [classname, classname1]]
     ```

4. Create the protocol provider:
   - Using Jacl:

     ```
     $AdminConfig create ProtocolProvider $newmp $ppAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('ProtocolProvider', newmp, ppAttrs)
     ```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#ProtocolProvider_4)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new custom properties using scripting

You can use scripting and the wsadmin tool to configure new custom properties.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property:

1. Identify the parent ID:
   - Using Jacl:

     ```
     set newmp [$AdminConfig getid /Cell:mycell/Node:mynode/MailProvider:MP1/]
     ```

   - Using Jython:

     ```
     newmp = AdminConfig.create('MailProvider', node, mpAttrs)
     print newmp
     ```

   Example output:

   ```
   MP1(cells/mycell/nodes/mynode|resources.xml#MailProvider_1)
   ```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newmp propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newmp, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_2)
   ```

3. Get required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute       Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name CP1]
     set cpAttrs [list $name]
     ```

     Example output:

     ```
     {name CP1}
     ```

   - Using Jython:

     ```
     name = ['name', 'CP1']
     cpAttrs = [name]
     print cpAttrs
     ```

Example output:

```
[[name, CP1]]
```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $cpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, cpAttrs)
     ```

   Example output:

   ```
   CP1(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_2)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new resource environment providers using scripting

You can use the wsadmin tool and scripting to configure new resources environment providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new resource environment provider:

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ResourceEnvironmentProvider
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ResourceEnvironmentProvider')
     ```

   Example output:

   ```
   Attribute     Type
   name     String
   ```

3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:

     ```
     set n1 [list name REP1]
     set repAttrs [list $name]
     ```

   - Using Jython:

     ```
     n1 = ['name', 'REP1']
     repAttrs = [n1]
     ```

4. Create a new resource environment provider:
   - Using Jacl:

     ```
     set newrep [$AdminConfig create ResourceEnvironmentProvider $node $repAttrs]
     ```

- Using Jython:

```
newrep = AdminConfig.create('ResourceEnvironmentProvider', node, repAttrs)
print newrep
```

Example output:

```
REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring custom properties for resource environment providers using scripting

You can use scripting to configure custom properties for a resource environment provider.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a resource environment provider:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

```
set newrep [$AdminConfig getid /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
```

   - Using Jython:

```
newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
print newrep
```

   Example output:

```
REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
```

2. Identify the required attributes:
   - Using Jacl:

```
$AdminConfig required J2EEResourceProperty
```

   - Using Jython:

```
print AdminConfig.required('J2EEResourceProperty')
```

   Example output:

```
Attribute     Type
name     String
```

3. Set up the required attributes and assign it to the repAttrs variable:
   - Using Jacl:

```
set name [list name RP]
set rpAttrs [list $name]
```

   - Using Jython:

```
name = ['name', 'RP']
rpAttrs = [name]
```

4. Get the J2EE resource property set:
   - Using Jacl:

```
set propSet [$AdminConfig showAttribute $newrep propertySet]
```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(newrep, 'propertySet')
print propSet
```

   Example output:

```
(cells/mycell/nodes/mynode|resources.xml#PropertySet_1)
```

If the command returns `None` as the value for the propSet variable, create a new property set. The command returns `None` if the property set does not exist in your environment. Use the following examples to create a new property set:

Using Jacl:

```
set newPropSet  [$AdminConfig  create  $newrep {}]
```

Using Jython:

```
newPropSet = AdminConfig.create('J2EEResourcePropertySet',newrep,[])
```

After setting the newPropSet variable, retry the command to get the J2EE resource property set before going to the next step.

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```
   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```
6. Save the configuration changes.

   Using Jacl:

   ```
   $AdminConfig save
   ```

   Using Jython:

   ```
   AdminConfig.save()
   ```

Configuring new referenceables using scripting

You can use scripting and the wsadmin tool to configure new referenceables.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new referenceable:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/
     ResourceEnvironmentProvider:REP1/]
     ```
   - Using Jython:

     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/
     ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:

   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```
2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required Referenceable
     ```
   - Using Jython:

     ```
     print AdminConfig.required('Referenceable')
     ```

Example output:

```
Attribute     Type
factoryClassname   String
classname   String
```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set fcn [list factoryClassname REP1]
     set cn [list classname NM1]
     set refAttrs [list $fcn $cn]
     ```

   - Using Jython:

     ```
     fcn = ['factoryClassname', 'REP1']
     cn = ['classname', 'NM1']
     refAttrs = [fcn, cn]
     print refAttrs
     ```

   Example output:

   ```
   {factoryClassname {REP1}} {classname {NM1}}
   ```

4. Create a new referenceable:
   - Using Jacl:

     ```
     set newref [$AdminConfig create Referenceable  $newrep  $refAttrs]
     ```

   - Using Jython:

     ```
     newref = AdminConfig.create('Referenceable',  newrep,  refAttrs)
     print newref
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#Referenceable_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new resource environment entries using scripting

You can use scripting and the wsadmin tool to configure a new resource environment entry.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new resource environment entry:

1. Identify the parent ID and assign it to the newrep variable.
   - Using Jacl:

     ```
     set newrep [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/]
     ```

   - Using Jython:

     ```
     newrep = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvironmentProvider:REP1/')
     print newrep
     ```

   Example output:

   ```
   REP1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvironmentProvider_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required ResourceEnvEntry
     ```

   - Using Jython:

     ```
     print AdminConfig.required('ResourceEnvEntry')
     ```

Example output:

```
Attribute     Type
name      String
jndiName   String
referenceable   Referenceable@
```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name REE1]
     set jndiName [list jndiName myjndi]
     set newref [$AdminConfig getid /Cell:mycell/Node:mynode/Referenceable:/]
     set ref [list referenceable $newref]
     set reeAttrs [list $name $jndiName $ref]
     ```

   - Using Jython:

     ```
     name = ['name', 'REE1']
     jndiName = ['jndiName', 'myjndi']
     newref = AdminConfig.getid('/Cell:mycell/Node:mynode/Referenceable:/')
     ref = ['referenceable', newref]
     reeAttrs = [name, jndiName, ref]
     ```

4. Create the resource environment entry:
   - Using Jacl:

     ```
     $AdminConfig create ResourceEnvEntry $newrep $reeAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('ResourceEnvEntry', newrep, reeAttrs)
     ```

   Example output:

   ```
   REE1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvEntry_1)
   ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring custom properties for resource environment entries using scripting

You can use scripting to configure a new custom property for a resource environment entry.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a resource environment entry:

1. Identify the parent ID and assign it to the newree variable.
   - Using Jacl:

     ```
     set newree [$AdminConfig  getid  /Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/]
     ```

   - Using Jython:

     ```
     newree = AdminConfig.getid('/Cell:mycell/Node:mynode/ResourceEnvEntry:REE1/')
     print newree
     ```

   Example output:

   ```
   REE1(cells/mycell/nodes/mynode|resources.xml#ResourceEnvEntry_1)
   ```

2. Create the J2EE custom property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newree propertySet]
     ```

   - Using Jython:

```
propSet = AdminConfig.showAttribute(newree, 'propertySet')
print propSet
```

Example output:

```
(cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_5)
```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name      String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP1]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP1']
     rpAttrs = [name]
     ```

5. Create the J2EE custom property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RPI(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring new URL providers using scripting

You can use scripting and the wsadmin tool to configure new URL providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new URL provider:

1. Identify the parent ID and assign it to the node variable.
   - Using Jacl:

     ```
     set node [$AdminConfig  getid  /Cell:mycell/Node:mynode/]
     ```

   - Using Jython:

     ```
     node = AdminConfig.getid('/Cell:mycell/Node:mynode/')
     print node
     ```

   Example output:

   ```
   mynode(cells/mycell/nodes/mynode|node.xml#Node_1)
   ```

2. Identify the required attributes:
   - Using Jacl:

```
$AdminConfig required URLProvider
```

- Using Jython:

```
print AdminConfig.required('URLProvider')
```

Example output:

```
Attribute     Type
streamHandlerClassName   String
protocol       String
name         String
```

3. Set up the required attributes:
   - Using Jacl:

```
set name [list name URLP1]
set shcn [list streamHandlerClassName "Put the stream handler classname here"]
set protocol [list protocol "Put the protocol here"]
set urlpAttrs [list $name $shcn $protocol]
```

Example output:

```
{name URLP1} {streamHandlerClassName {Put the stream handler classname here}}
{protocol {Put the protocol here}}
```

   - Using Jython:

```
name = ['name', 'URLP1']
shcn = ['streamHandlerClassName', "Put the stream handler classname here"]
protocol = ['protocol', "Put the protocol here"]
urlpAttrs = [name, shcn, protocol]
print urlpAttrs
```

Example output:

```
[[name, URLP1], [streamHandlerClassName, "Put the stream handler classname here"],
[protocol, "Put the protocol here"]]
```

4. Create a URL provider:
   - Using Jacl:

```
$AdminConfig create URLProvider $node $urlpAttrs
```

   - Using Jython:

```
print AdminConfig.create('URLProvider', node, urlpAttrs)
```

Example output:

```
URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring custom properties for URL providers using scripting

You can use scripting to configure custom properties for URL providers.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure custom properties for URL providers:

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:

```
set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
```

   - Using Jython:

```
newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
print newurlp
```

Example output:

```
URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
```

2. Get the J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newurlp propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newurlp, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#PropertySet_7)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP2]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP2']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP2(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_1)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Related tasks

“Configuring custom properties for URLs using scripting” on page 2040
Use the wsadmin tool and scripting to set custom properties for URLs.

Configuring new URLs using scripting

You can use scripting and the wsadmin tool to configure new URLs.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following example to configure a new URL:

1. Identify the parent ID and assign it to the newurlp variable.
   - Using Jacl:

     ```
     set newurlp [$AdminConfig  getid  /Cell:mycell/Node:mynode/URLProvider:URLP1/]
     ```

   - Using Jython:

     ```
     newurlp = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/')
     print newurlp
     ```

     Example output:

     ```
     URLP1(cells/mycell/nodes/mynode|resources.xml#URLProvider_1)
     ```

2. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required URL
     ```

   - Using Jython:

     ```
     print AdminConfig.required('URL')
     ```

     Example output:

     ```
     Attribute     Type
     name          String
     spec          String
     ```

3. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name URL1]
     set spec [list spec "Put the spec here"]
     set urlAttrs [list $name $spec]
     ```

     Example output:

     ```
     {name URL1} {spec {Put the spec here}}
     ```

   - Using Jython:

     ```
     name = ['name', 'URL1']
     spec = ['spec', "Put the spec here"]
     urlAttrs = [name, spec]
     ```

     Example output:

     ```
     [[name, URL1], [spec, "Put the spec here"]]
     ```

4. Create a URL:
   - Using Jacl:

     ```
     $AdminConfig create URL $newurlp $urlAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('URL', newurlp, urlAttrs)
     ```

     Example output:

     ```
     URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
     ```

5. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Configuring custom properties for URLs using scripting

Use the wsadmin tool and scripting to set custom properties for URLs.

Before you begin

Before starting this task, the wsadmin tool must be running. See the Starting the wsadmin scripting client article for more information.

About this task

Perform the following steps to configure a new custom property for a URL:

1. Identify the parent ID and assign it to the newurl variable.
   - Using Jacl:

     ```
     set newurl [$AdminConfig getid /Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/]
     ```

   - Using Jython:

     ```
     newurl = AdminConfig.getid('/Cell:mycell/Node:mynode/URLProvider:URLP1/URL:URL1/')
     print newurl
     ```

   Example output:

   ```
   URL1(cells/mycell/nodes/mynode|resources.xml#URL_1)
   ```

2. Create a J2EE resource property set:
   - Using Jacl:

     ```
     set propSet [$AdminConfig showAttribute $newurl propertySet]
     ```

   - Using Jython:

     ```
     propSet = AdminConfig.showAttribute(newurl, 'propertySet')
     print propSet
     ```

   Example output:

   ```
   (cells/mycell/nodes/mynode|resources.xml#J2EEResourcePropertySet_7)
   ```

3. Identify the required attributes:
   - Using Jacl:

     ```
     $AdminConfig required J2EEResourceProperty
     ```

   - Using Jython:

     ```
     print AdminConfig.required('J2EEResourceProperty')
     ```

   Example output:

   ```
   Attribute     Type
   name         String
   ```

4. Set up the required attributes:
   - Using Jacl:

     ```
     set name [list name RP3]
     set rpAttrs [list $name]
     ```

   - Using Jython:

     ```
     name = ['name', 'RP3']
     rpAttrs = [name]
     ```

5. Create a J2EE resource property:
   - Using Jacl:

     ```
     $AdminConfig create J2EEResourceProperty $propSet $rpAttrs
     ```

   - Using Jython:

     ```
     print AdminConfig.create('J2EEResourceProperty', propSet, rpAttrs)
     ```

   Example output:

   ```
   RP3(cells/mycell/nodes/mynode|resources.xml#J2EEResourceProperty_7)
   ```

6. Save the configuration changes. See the Saving configuration changes with the wsadmin tool article for more information.

Provider command group for the AdminTask object

You can use the Jython or Jacl scripting languages to configure mail settings with the wsadmin tool. The commands and parameters in the provider group can be used to create, delete, and manage providers.

The Provider command group for the AdminTask object includes the following commands:

deleteProvider

The **deleteProvider** command deletes a provider from the model given the providerName.

Target object

None

Parameters and return values
- Parameters: None
- Returns: true if the provider was deleted, false if not.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask deleteProvider {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.deleteProvider ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.deleteProvider (['-interactive'])
  ```

getProviderInfo

The **getProviderInfo** command returns a ProviderInfo object defined in the model given the providerName.

Target object

None

Parameters and return values
- Parameters: None
- Returns: A ProviderInfo object defined in the model given the providerName.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask getProviderInfo {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviderInfo ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviderInfo (['-interactive'])
  ```

getProviderInstance

The **getProviderInstance** command returns a java.security.Provider for the providerName specified.

Target object

None

Parameters and return values
- Parameters: None
- Returns: A java.security.Provider for the providerName specified.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask getProviderInsta nce {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviderIns tance ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviderInst ance (['-interactive'])
  ```

getProviders

The **getProviders** command returns a List of ProviderInfo objects from the model.

Target object

None

Parameters and return values
- Parameters: None
- Returns: A list of ProviderInfo objects from the model.

Examples

Interactive mode example usage:
- Using Jacl:
  ```
  $AdminTask getProviders {-interactive}
  ```
- Using Jython string:
  ```
  AdminTask.getProviders ('[-interactive]')
  ```
- Using Jython list:
  ```
  AdminTask.getProviders (['-interactive'])
  ```

Related tasks

Using the AdminTask object for scripted administration
Use the AdminTask object to access a set of administrative commands that provide an alternative way to access the configuration commands and the running object management commands.

Related reference

Commands for the AdminTask object
Use the AdminTask object to run administrative commands with the wsadmin tool.

Chapter 10. Security (omitted)

This PDF book corresponds closely to the **Administering WebSphere applications** section of the online information center. However, the security chapter of this book has been omitted, because the security information is available in the PDF book for **Securing WebSphere applications and their environment**.

Chapter 11. Naming and directory

Using naming

Naming is used by clients of WebSphere Application Server applications most commonly to obtain references to objects related to those applications, such as Enterprise JavaBeans (EJB) homes.

About this task

The Naming service is based on the Java Naming and Directory Interface (JNDI) Specification and the Object Management Group (OMG) Interoperable Naming (CosNaming) specifications Naming Service Specification, Interoperable Naming Service revised chapters and Common Object Request Broker: Architecture and Specification (CORBA).

1. Develop your application using either JNDI or CORBA CosNaming interfaces. Use these interfaces to look up server application objects that are bound into the namespace and obtain references to them. Most Java developers use the JNDI interface. However, the CORBA CosNaming interface is also available for performing Naming operations on WebSphere Application Server name servers or other CosNaming name servers.
2. Assemble your application using an assembly tool. Application assembly is a packaging and configuration step that is a prerequisite to application deployment. If the application you are assembling is a client to an application running in another process, you should qualify the jndiName values in the deployment descriptors for the objects related to the other application. Otherwise, you might need to override the names with qualified names during application deployment. If the objects have fixed qualified names configured for them, you should use them so that the jndiName values do not depend on the other application's location within the topology of the cell.
3. Optional: Verify that your application is assigned the appropriate security role if administrative security is enabled. For more information on the security roles, see "Naming roles" on page 2056.
4. Deploy your application. Put your assembled application onto the application server. If the application you are assembling is a client to an application running in another server process, be sure to qualify the jndiName values for the other application's server objects if they are not already qualified. For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 2051.
5. Configure namespace bindings. This step is necessary in these cases:
   - Your deployed application is to be accessed by legacy client applications running on previous versions of the product. In this case, you must configure additional name bindings for application objects relative to the default initial context for legacy clients. (Version 5 clients have a different initial context from legacy clients.)
   - The application requires qualified name bindings for such reasons as:
     – It will be accessed by Java Platform, Enterprise Edition (Java EE) client applications or server applications running in another server process.
     – It will be accessed by thin client applications.

     In this case, you can configure name bindings as additional bindings for application objects. The qualified names for the configured bindings are *fixed*, meaning they do not contain elements of the cell topology that can change if the application is moved to another server. Objects as bound into the namespace by the system can always be qualified with a topology-based name. You must explicitly configure a name binding to use as a fixed qualified name.

   For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 2051. For more information on configured name bindings, refer to "Configured name bindings" on page 2054.
6. Troubleshoot any problems that develop. If a Naming operation is failing and you need to verify whether certain name bindings exist, use the dumpNameSpace tool to generate a dump of the namespace.

Naming

Naming is used by clients of WebSphere Application Server applications to obtain references to objects related to those applications, such as enterprise bean (EJB) homes.

These objects are bound into a mostly hierarchical structure, referred to as a *namespace*. In this structure, all non-leaf objects are called *contexts*. Leaf objects can be contexts and other types of objects. Naming operations, such as lookups and binds, are performed on contexts. All naming operations begin with obtaining an *initial context*. You can view the initial context as a starting point in the namespace.

The namespace structure consists of a set of *name bindings*, each consisting of a name relative to a specific context and the object bound with that name. For example, the name `myApp/myEJB` consists of one non-leaf binding with the name `myApp`, which is a context. The name also includes one leaf binding with the name `myEJB`, relative to `myApp`. The object bound with the name `myEJB` in this example happens to be an EJB home reference. The whole name `myApp/myEJB` is relative to the initial context, which you can view as a starting place when performing naming operations.

You can access and manipulate the namespace through a *name server*. Users of a name server are referred to as *naming clients*. Naming clients typically use the Java Naming and Directory Interface (JNDI) to perform naming operations. Naming clients can also use the Common Object Request Broker Architecture (CORBA) CosNaming interface.

You can use security to control access to the namespace. For more information, see *Naming roles*.

Typically, objects bound to the namespace are resources and objects associated with installed applications. These objects are bound by the system, and client applications perform lookup operations to obtain references to them. Occasionally, server and client applications bind objects to the namespace. An application can bind objects to transient or persistent partitions, depending on requirements.

In Java Platform, Enterprise Edition (Java EE) or Java Platform, Standard Edition (Java SE) environments, some JNDI operations are performed with `java:` URL names. Names bound under these names are bound to a completely different namespace which is local to the calling process. However, some lookups on the `java:` namespace may trigger indirect lookups to the name server.

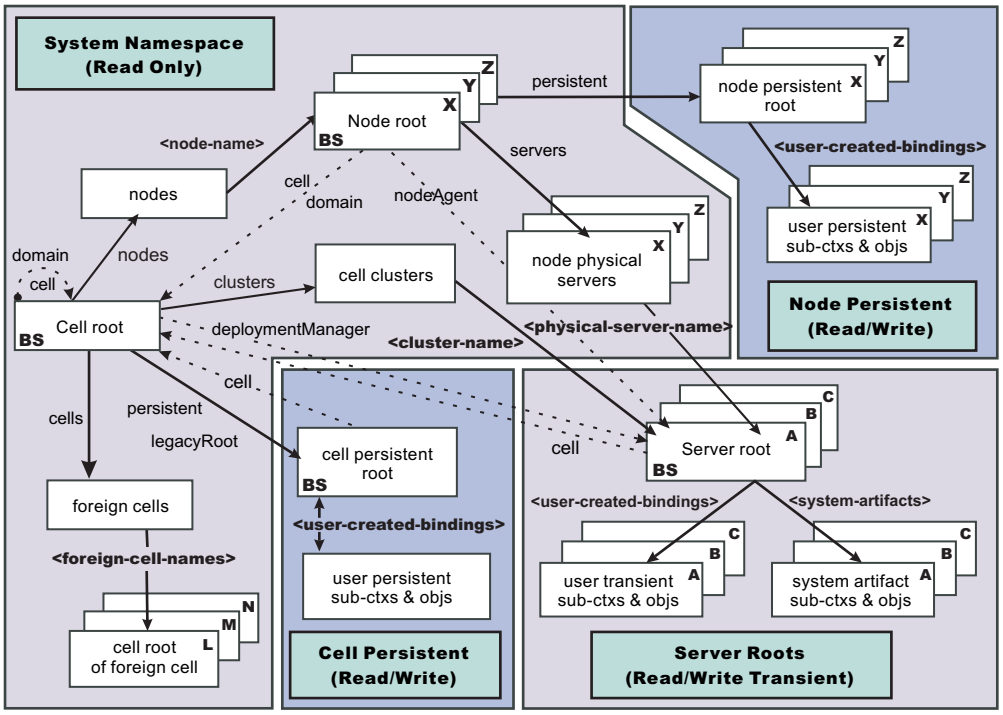
Namespace logical view

The namespace for the entire cell is federated among all servers in the cell. Every server process contains a name server. All name servers provide the same logical view of the cell namespace.

The various server roots and persistent partitions of the namespace are interconnected by a system namespace. You can use the system namespace structure to traverse to any context in a cell's namespace.

A logical view of the namespace in a multiple-server installation is shown in the following diagram.

Logical View of a Cell's Namespace



The bindings in the preceding diagram appear with solid arrows, labeled in bold, and dashed arrows, labeled in gray. Solid arrows represent *primary bindings*. A primary binding is formed when the associated subcontext is created. Dashed arrows show *linked bindings*. A linked binding is formed when an existing context is bound under an additional name. Linked bindings are added for convenience or interoperability with previous WebSphere Application Server versions.

A cell namespace is composed of contexts which reside in servers throughout the cell. All name servers in the cell provide the same logical view of the cell namespace. A name server constructs this view at startup by reading configuration information. Each name server has its own local in-memory copy of the namespace and does not require another running server to function. There are, however, a few exceptions. Server roots for other servers are not replicated among all the servers. The respective server for a server root must be running to access that server root context.

Namespace partitions

There are four major partitions in a cell namespace:
- System namespace partition
- Server roots partition
- Cell persistent partition
- Node persistent partition

System namespace partition

The system namespace contains a structure of contexts based on the cell topology. The system structure supports traversal to all parts of a cell namespace and to the cell root of other cells, which are configured as foreign cells. The root of this structure is the cell root. In addition to the cell root, the system structure contains a node root for each node in the cell. You can access other contexts of interest specific to a node from the node root, such as the node persistent root and server roots for servers configured in that node.

All contexts in the system namespace are read-only. You cannot add, update, or remove any bindings.

Server roots partition

Each server in a cell has a server root context. A server root is specific to a particular server. You can view the server roots for all servers in a cell as being in a transient read/write partition of the cell namespace. System artifacts, such as enterprise bean (EJB) homes for server applications and resources, are bound under the server root context of the associated server. A server application can also add bindings under its server root. These bindings are transient. Therefore, the server application creates all required bindings at application startup, so they exist anytime the application is running.

Server-scoped configured name bindings are relative to a server's server root.

Cell persistent partition

The root context of the cell persistent partition is the cell persistent root. A binding created under the cell persistent root is saved as part of the cell configuration and continues to exist until it is explicitly removed. Applications that need to create additional persistent bindings of objects generally associated with the cell can bind these objects under the cell persistent root.

It is important to note that the cell persistent area is not designed for transient, rapidly changing bindings. The bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Cell-scoped configured name bindings are relative to a cell's cell persistent root.

Node persistent partition

The node persistent partition is similar to the cell partition except that each node has its own node persistent root. A binding created under a node persistent root is saved as part of that node configuration and continues to exist until it is explicitly removed.

Applications that need to create additional persistent bindings of objects associated with a specific node can bind those objects under that particular node's node persistent root. As with the cell persistent area, it is important to note that the node persistent area is not designed for transient, rapidly changing bindings. These bindings are more static in nature, such as part of an application setup or configuration, and are not created at run time.

Node-scoped configured name bindings are relative to a node's node persistent root.

Initial context support

All naming operations begin with obtaining an initial context. You can view the initial context as a starting point in the namespace. Use the initial context to perform naming operations, such as looking up and binding objects in the namespace.

Initial contexts registered with the ORB as initial references

The server root, cell persistent root, cell root, and node root are registered with the name server's ORB and can be used as an initial context. An initial context is used by CORBA and enterprise bean applications as a starting point for namespace lookups. The keys for these roots as recognized by the ORB are shown in the following table:

Root Context	Initial Reference Key
Server Root	NameServiceServerRoot
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot, NameService
Node Root	NameServiceNodeRoot

A server root initial context is the server root context for the specific server you are accessing. Similarly, a node root initial context is the node root for the server being accessed.

You can use the previously mentioned keys in CORBA INS object URLs (corbaloc and corbaname) and as an argument to an ORB `resolve_initial_references` call. For examples, see CORBA and JNDI programming examples, which show how to get an initial context.

Default initial contexts

The default initial context depends on the type of client. Different categories of clients and the corresponding default initial context follow.

- **WebSphere Application Server V5 and later JNDI interface implementation**

  The JNDI interface is used by EJB applications to perform namespace lookups. WebSphere Application Server clients by default use the WebSphere Application Server CosNaming JNDI plug-in implementation. The default initial context for clients of this type is the server root of the server specified by the provider URL. For more details, refer to the JNDI programming examples on getting initial contexts.
- **Other JNDI implementation**

  Some applications can perform namespace lookups with a non-product CosNaming JNDI plug-in implementation. Assuming the key **`NameService`** is used to obtain the initial context, the default initial context for clients of this type is the cell root.
- **CORBA**

  The standard CORBA client obtains an initial org.omg.CosNaming.NamingContext reference with the key **`NameService`**. The initial context in this case is the cell root.

Lookup names support in deployment descriptors and thin clients

Server application objects, such as enterprise bean (EJB) homes, are bound relative to the server root context for the server in which the application is installed. Other objects, such as resources, can also be bound to a specific server root. The names used to look up these objects must be qualified so as to select the correct server root. This topic discusses what relative and qualified names are, when they can be used, and how you can construct them.

Relative names

All names are relative to a context. Therefore, a name that can be resolved from one context in the namespace cannot necessarily be resolved from another context in the namespace. This point is significant because the system binds objects with names relative to the server root context of the server in which the application is installed. Each server has its own server root context. The initial Java Naming and Directory Interface (JNDI) context is by default the server root context for the server identified by the provider URL used to obtain the initial context. (Typically, the URL consists of a host and port.) For applications running in a server process, the default initial JNDI context is the server root for that server. A relative name will resolve successfully when the initial context is obtained from the server which contains the target object, but it will not resolve successfully from an initial context obtained from another server.

If all clients of a server application run in the same server process as the application, all objects associated with that application are bound to the same initial context as the clients' initial context. In this case, only names relative to the server's server root context are required to access these server objects. Frequently, however, a server application has clients that run outside the application's server process. The initial context for these clients can be different from the server application's initial context, and lookups on the relative names for server objects may fail. These clients need to use the qualified name for the server objects. This point must be considered when setting up the jndiName values in a Java Platform, Enterprise Edition (Java EE) client application deployment descriptors and when constructing lookup names in thin clients. Qualified names resolve successfully from any initial context in the cell.

Qualified names

All names are relative to a context. Here, the term *qualified name* refers to names that can be resolved from any initial context in a cell. This action is accomplished by using names that navigate to the same context, the cell root. The rest of the qualified name is then relative to the cell root and uniquely identifies an object throughout the cell. All initial contexts in a server (that is, all naming contexts in a server registered with the ORB as an initial reference) contain a binding with the name **cell**, which links back to the cell root context. All qualified names begin with the string **cell/** to navigate from the current initial context back to the cell root context.

A qualified name for an object is the same throughout the cell. The name can be topology-based, or some fixed name bound under the cell persistent root. Topology-based names, described in more detail below, navigate through the system namespace to reach the target object. A fixed name bound under the cell persistent root has the same qualified name throughout the cell and is independent of the topology. Creating a fixed name under the cell persistent root for a server application object requires an extra step when the server application is installed, but this step eliminates impacts to clients when the application is moved to a different location in the cell topology. The process for creating a fixed name is described later in this section.

Generally, you **must** use qualified names for EJB jndiName values in a Java EE client application deployment descriptors and for EJB lookup names in thin clients. The only exception is when the initial context is obtained from the server in which the target object resides. For example, a session bean which is a client to an entity bean can use a relative name if the two beans run in the same server. If the session bean and entity beans run in different servers, the jndiName for the entity bean must be qualified in the session bean's deployment descriptors. The same requirement may be true for resources as well, depending on the scope of the resource.

- **Topology-based names**

  The system namespace partition in a cell's namespace reflects the cell's topology. This structure can be navigated to reach any object bound into the cell's namespace. Topology-based qualified names include elements from the topology which reflect the object's location within the cell.

  Single server

  An object bound in a single server has a topology-based qualified name of the following form:

  `cell/nodes/`*`nodeName`*`/servers/`*`serverName`*`/`*`relativeJndiName`*

  where *nodeName* and *serverName* are the node name and server name for the server where the object is bound, and *relativeJndiName* is the unqualified name of the object; that is, the object's name relative to its server's server root context.

- **Fixed names**

  It is possible to create a fixed name for a server object so that the qualified name is independent of the cell topology. This quality is desirable when clients of the application run in other server processes or as pure clients. Fixed names have the advantage of not changing if the object is moved to another server. The jndiName values in deployment descriptors for a Java EE client application can reference the qualified fixed name for a server object regardless of the cell topology on which the client or server application is being installed.

  Defining a cell-wide fixed name for a server application object requires an extra step after the server application is installed. That is, a binding for the object must be created under the cell persistent root. A fixed name bound under the cell persistent root can be any name, but all names under the cell persistent root must be unique within the cell because the cell persistent root is global to the entire cell.

  A qualified fixed name has the form:

  `cell/persistent/`*`fixedName`*

  where *fixedName* is an arbitrary fixed name.

  The binding can be created programmatically (for example, using JNDI). However, it is probably more convenient to configure a cell-scoped binding for the server object.

You must keep the programmatic or configured binding up-to-date. Configured EJB bindings are based on the location of the enterprise bean within the cell topology, and moving the EJB application to another server, for example, requires the configured binding to be updated. Similar changes affect an EJB home reference programmatically bound so that the fixed name would need to be rebound with a current reference. However, for Java EE clients, the jndiName value for the object, and for thin clients, the lookup name for the object, remains the same. In other words, clients that access objects by fixed names are not affected by changes to the configuration of server applications they access.

Using lookup names in deployment descriptor bindings

Java EE applications can contain deployment descriptors, such as ejb-ref, resource-ref, and resource-env-ref, that are used to declare various types of references. These reference declarations define java:comp/env lookup names that are available to corresponding Java EE components. Each java:comp/env lookup name must be mapped to a lookup name in the global name space, relative to the server root context, which is the default initial JNDI context.

If a reference maps to an object that is bound under the server root context for the same server as the component that is executing the lookup, you can use a relative lookup name. If a reference maps to an object that is bound under the server root context of another server, you must qualify the lookup name. For example, you must qualify a lookup name if a servlet, that is running on one server, declares an ejb-ref for an EJB that is running on another server. Similarly, if the reference maps to an object that is bound into a persistent partition of the name space, or to an object that is bound through a cell-scoped or node-scoped configured name space binding, you must use a qualified name.

You can specify deployment descriptor reference binding values when you install the application, and edit them after the application is installed. If you need to change the JNDI lookup name a reference maps to, in the administrative console, click **Applications > Enterprise Applications >** *application_name*. In the References section, there are links that correspond to the various reference types, such as EJB references and Resource environment entry references, that are declared by this application. Click on the link for the reference type that you need to change, and then specify a new value in the **Target Resource JNDI Name** field.

JNDI support in WebSphere Application Server

The product includes a name server to provide shared access to Java components, and an implementation of the `javax.naming` JNDI package which supports user access to the name server through the Java Naming and Directory Interface (JNDI) naming interface.

WebSphere Application Server does ***not*** provide implementations for:
- `javax.naming.directory` or
- `javax.naming.ldap` packages

Also, WebSphere Application Server does ***not*** support interfaces defined in the `javax.naming.event` package.

However, to provide access to LDAP servers, the development kit shipped with WebSphere Application Server supports the Sun Microsystems implementation of:
- `javax.naming.ldap` and
- `com.sun.jndi.ldap.LdapCtxFactory`

The WebSphere Application Server JNDI implementation is based on the JNDI interface, and was tested with the Sun Microsystems JNDI Service Provider Interface (SPI).

The default behavior of this JNDI implementation is adequate for most users. However, users with specific requirements can control certain aspects of JNDI behavior.

Configured name bindings

Administrators can configure bindings into the namespace. A configured binding is different from a programmatic binding in that the system creates the binding every time a server is started, even if the target context is in a transient partition.

Administrators can add name bindings to the namespace through the configuration. Name servers add these configured bindings to the namespace view, by reading the configuration data for the bindings. Configuring bindings is an alternative to creating the bindings from a program. Configured bindings have the advantage of being created each time a server starts, even when the binding is created in a transient partition of the namespace. Cell-scoped configured bindings provide a fixed qualified name for server application objects.

Scope

You can configure a binding at one of the following four scopes: cell, node, server, or cluster. Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server. Cluster-scoped bindings are created under the server root context in each member of the selected cluster.

The scope you select for new bindings depends on how the binding is to be used. For example, if the binding is not specific to any particular node, cluster, or server, or if you do not want the binding to be associated with any specific node, cluster, or server, a cell-scoped binding is a suitable scope. Defining fixed names for enterprise beans to create fixed qualified names is just such an application. If a binding is to be used only by clients of an application running on a particular server (or cluster), or if you want to configure a binding with the same name on different servers (or clusters) which resolve to different objects, a server-scoped (or cluster-scoped) binding would be appropriate. Note that two servers or clusters can have configured bindings with the same name but resolve to different objects. At the cell scope, only one binding with a given name can exist.

Intermediate contexts

Intermediate contexts created with configured bindings are read-only. For example, if an EJB home binding is configured with the name `some/compound/name/ejbHome`, the intermediate contexts `some`, `some/compound`, and `some/compound/name` will be created as read-only contexts. You cannot add, update, or remove any read-only bindings.

The configured binding name cannot conflict with existing bindings. However, configured bindings can use the same intermediate context names. Therefore, a configured binding with the name `some/compound/name2/ejbHome2` does not conflict with the previous example name.

Configured binding types

Types of objects that you can bind follow:

EJB: EJB home installed in some server in the cell

The following data is required to configure an EJB home binding:

- JNDI name of the EJB server or server cluster where the enterprise bean is deployed
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped EJB binding is useful for creating a fixed lookup name for an enterprise bean so that the qualified name is not dependent on the topology.

Note: In standalone servers, an EJB binding resolving to another server cannot be configured because the name server does not read configuration data for other servers. That data is required to construct the binding.

CORBA: CORBA object available from some CosNaming name server

You can identify any CORBA object bound into some INS compliant CosNaming server with a corbaname URL. The referenced object does not have to be available until the binding is actually referenced by some application.

The following data is required in order to configure a CORBA object binding:

- The corbaname URL of the CORBA object
- An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object)
- Target root for the configured binding
- The name of the configured binding, relative to the target root

Indirect: Any object bound in WebSphere Application Server namespace accessible with JNDI

Besides CORBA objects, this includes javax.naming.Referenceable, javax.naming.Reference, and java.io.Serializable objects. The target object itself is not bound to the namespace. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. The following data is required in order to configure an indirect JNDI lookup binding:

- JNDI provider URL of name server where object resides
- JNDI lookup name of object
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root.

A cell-scoped indirect binding is useful when creating a fixed lookup name for a resource so that the qualified name is not dependent on the topology. You can also achieve this topology by widening the scope of the resource definition.

String: String constant

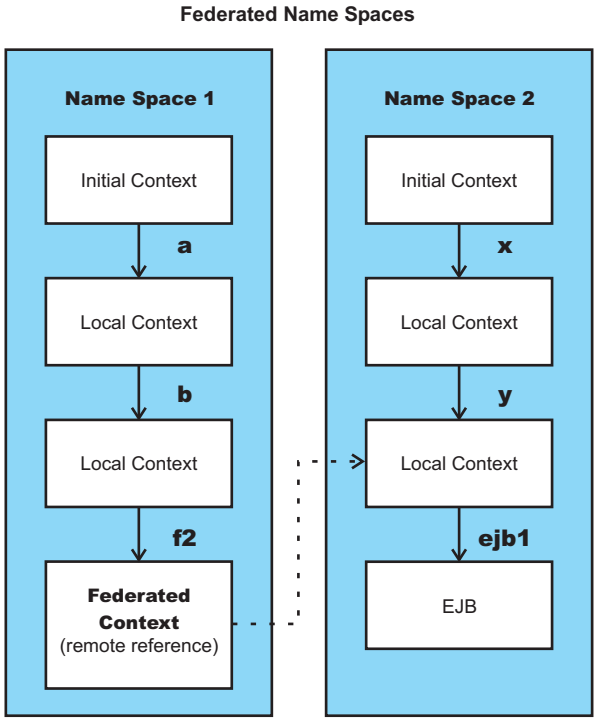
You can configure a binding of a string constant. The following data is required to configure a string constant binding:

- String constant value
- Target root for the configured binding (scope)
- The name of the configured binding, relative to the target root

Namespace federation

Federating namespaces involves binding contexts from one namespace into another namespace.

For example, assume that a namespace, Namespace 1, contains a context under the name `a/b`. Also assume that a second namespace, Namespace 2, contains a context under the name `x/y`. (See the following illustration.) If context `x/y` in Namespace 2 is bound into context `a/b` in Namespace 1 under the name `f2`, the two namespaces are federated. Binding `f2` is a federated binding because the context associated with that binding comes from another namespace. From Namespace 1, a lookup of the name `a/b/f2` returns the context bound under the name `x/y` in Namespace 2. Furthermore, if context `x/y` contains an enterprise bean (EJB) home bound under the name `ejb1`, the EJB home can be looked up from Namespace 1 with the lookup name `a/b/f2/ejb1`. Notice that the name crosses namespaces. This fact is transparent to the naming client.



In a product namespace, you can create federated bindings with the following restrictions:

- Federation is limited to CosNaming name servers. A product name server is a Common Object Request Broker Architecture (CORBA) CosNaming implementation. You can create federated bindings to other CosNaming contexts. You cannot, for example, bind contexts from an LDAP name server implementation.
- If you use JNDI to federate the namespace, you must use a WebSphere Application Server initial context factory to obtain the reference to the federated context. If you use some other initial context factory implementation, you might not be able to create the binding or the level of transparency might be reduced.
- A federated binding to a non-product naming context has the following functional limitations:
  - JNDI operations are restricted to the use of CORBA objects. For example, you can look up EJB homes, but you cannot look up non-CORBA objects such as data sources.
  - JNDI caching is not supported for non-product namespaces. This restriction affects the performance of lookup operations only.
  - If security is enabled, the producr does not support federated bindings to non-product namespaces.
- Do not federate two product standalone server namespaces. Incorrect behavior might result. If you want to federate product namespaces, use servers running under the Network Deployment package of WebSphere Application Server.
- When federating the namespaces of two cells running a Network Deployment package of WebSphere Application Server, the names of the cells must be different. Otherwise, incorrect behavior can result.

Naming roles

The Java 2 Platform, Enterprise Edition (J2EE) role-based authorization concept is extended to protect the CosNaming service.

CosNaming security offers increased granularity of security control over CosNaming functions. CosNaming functions are available on CosNaming servers such as the WebSphere Application Server. They affect the content of the name space. Generally two ways are acceptable in which client programs result in

CosNaming calls. The first is through the Java Naming and Directory Interface (JNDI) methods. The second is CORBA clients invoking CosNaming methods directly.

The following security roles exist. However, the roles have an authority level from low to high as shown in the following list. The list also provides the security-related interface methods for each role. The interface methods that are not listed are either not supported or not relevant to security.

- **CosNamingRead**. Users who are assigned the CosNamingRead role can do queries of the name space, such as through the JNDI lookup method. The Everyone special-subject is the default policy for this role.

Table 34. CosNamingRead role packages and interface methods

Package	Interface methods
javax.naming	• Context.list • Context.listBindings • Context.lookup • NamingEnumeration.hasMore • NamingEnumeration.next
org.omg.CosNaming	• NamingContext.list • NamingContext.resolve • BindingIterator.next_one • BindingIterator.next_n • BindingIterator.destroy

- **CosNamingWrite**. Users who are assigned the CosNamingWrite role can do write operations (such as JNDI bind, rebind, or unbind) plus CosNamingRead operations. As a default policy, Subjects are not assigned this role.

Table 35. CosNamingWrite role packages and interface methods

Package	Interface methods
javax.naming	• Context.bind • Context.rebind • Context.rename • Context.unbind
org.omg.CosNaming	• NamingContext.bind • NamingContext.bind_context • NamingContext.rebind • NamingContext.rebind_context • NamingContext.unbind

- **CosNamingCreate**. Users who are assigned the CosNamingCreate role are allowed to create new objects in the name space through JNDI createSubcontext operations plus CosNamingWrite operations. As a default policy, Subjects are not assigned this role.

Table 36. CosNamingCreate role packages, interface methods

Package	Interface methods
javax.naming	Context.createSubcontext
org.omg.CosNaming	NamingContext.bind_new_context

- **CosNamingDelete**. Users who are assigned the CosNamingDelete role can destroy objects in the name space, for example byusing the JNDI destroySubcontext method and CosNamingCreate operations. As a default policy, Subjects are not assigned this role.

Table 37. CosNamingDelete role packages and interface methods

Package	Interface methods
javax.naming	Context.destroySubcontext

Table 37. CosNamingDelete role packages and interface methods (continued)

Package	Interface methods
org.omg.CosNaming	NamingContext.destroy

Note: The `javax.naming` package applies to the CosNaming JNDI service provider only. All of the variants of a JNDI interface method have the same role mapping.

If the caller is not authorized, the packages listed in the previous tables exhibit the following behavior:

javax.naming
This package creates the javax.naming.NoPermissionException exception, which maps NO_PERMISSION from the CosNaming method invocation to NoPermissionException.

org.omg.CosNaming
This package creates the org.omg.CORBA.NO_PERMISSION exception.

Users, groups, or the AllAuthenticated and Everyone special subjects can be added or removed to or from the naming roles from the WebSphere Application Server administrative console at any time. However, you must restart the server for the changes to take effect. A best practice is to map groups or one of the special-subjects, rather than specific users, to Naming roles because it is more flexible and easier to administer in the long run. By mapping a group to a naming role, adding or removing users to or from the group occurs outside of WebSphere Application Server and does not require a server restart for the change to take effect.

If a user is assigned a particular naming role and that user is a member of a group that is assigned a different naming role, the user is granted the most permissive access between the role that is assigned and the role the group is assigned. For example, assume that the MyUser user is assigned the CosNamingRead role. Also, assume that the MyGroup group is assigned the CosNamingCreate role. If the MyUser user is a member of the MyGroup group, the MyUser user is assigned the CosNamingCreate role because the user is a member of the MyGroup group. If the MyUser user is not a member of the MyGroup group, is assigned the CosNamingRead role.

The CosNaming authorization policy is only enforced when administrative security is enabled. When administrative security is enabled, attempts to do CosNaming operations without the proper role assignment result in a org.omg.CORBA.NO_PERMISSION exception from the CosNaming server.

In WebSphere Application Server, each CosNaming function is assigned to one role only. Therefore, users who are assigned the CosNamingCreate role cannot query the name space unless they also are assigned the CosNamingRead role. In most cases, a creator needs three roles assigned: CosNamingRead, CosNamingWrite, and CosNamingCreate. The CosNamingRead and CosNamingWrite roles assignment for the creator example in above have been included in CosNamingCreate role. In most cases, WebSphere Application Server administrators do not have to change the roles assignment for every user or group when they move to this release from a previous one.

Although the ability exists to greatly restrict access to the name space by changing the default policy, doing so might result in unexpected org.omg.CORBA.NO_PERMISSION exceptions at runtime. Typically, J2EE applications access the name space and the identity is that of the user that authenticated to WebSphere Application Server when he J2EE application is accessed. Unless the J2EE application provider clearly communicates the expected naming roles, fully consider changing the default naming authorization policy.

Naming and directories: Resources for learning

Additional information and guidance on naming and directories is available on various Internet sites.

Use the following links to find relevant supplemental information about naming and directories. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

The naming service provided with WebSphere Application Server Versions 6.x and 7.0 is the same as that provided for Version 5, thus information on the Version 5.0 naming and directories applies to Versions 6.x and 7.0.

The following links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Programming instructions and examples

- Naming in WebSphere Application Server V5: Impact on Migration and Interoperability, http://www.ibm.com/developerworks/websphere/library/techarticles/0305_weiner/weiner.html
- *WebSphere Application Server V6.1:* System Management Configuration Handbook, SG24-7304-00, http://www.redbooks.ibm.com/abstracts/SG247304.html?Open
- *IBM WebSphere Developer Technical Journal:* Co-hosting multiple versions of J2EE applications, http://www.ibm.com/developerworks/websphere/techjournal/0405_poddar/0405_poddar.html

Programming specifications

- Specifications and API documentation

Configuring name servers

Name servers add configured name space bindings to the name space view by reading the configuration data for the bindings. If you use configured bindings, you do not need to create name space bindings from a program.

About this task

You can configure a name server to provide a display name and initial state for an application server, as well as specify custom properties for the name server. Configure a name server using the administrative console.

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Administration** → **Server components** → **Name server**.
2. Edit the fields as desired.

   All of the fields are mandatory.
3. To make other changes, click **Custom properties** and configure a custom property.
4. Click **OK** to register your changes.

Name server settings

Use this page to configure Naming Service Provider settings for the application server.

To view this administrative console page, click one of the following paths:

- **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Administration** → **Server components** → **Name server**
- **Servers** → **Server Types** → **Version 5 JMS servers** → ***server_name*** → **Administration** → **Server components** → **Name server**

Name

Specifies the display name for the server component.

Data type String

Initial state
Specifies the execution state. The options are: *Started* and *Stopped*.

Data type String
Default Started

Configuring namespace bindings

Instead of creating namespace bindings from a program, you can configure namespace bindings using the administrative console. Name servers add these configured bindings to the namespace view by reading the configuration data for the bindings. Configured bindings are created each time a server starts, even when the binding is created in a transient partition of the namespace. One major use of configured bindings is to provide fixed qualified names for server application objects.

Before you begin

Assemble and deploy your application onto an application server. If the application is a client to an application running in another server process, specify qualified jndiName values for the server objects of the other application during assembly or deployment. For more information on qualified names, refer to "Lookup names support in deployment descriptors and thin clients" on page 2051.

About this task

A deployed application requires qualified fixed names if the application is accessed by thin client applications or by Java Platform, Enterprise Edition (Java EE) client applications or server applications running in another server process.

When you configure a namespace binding, you create a qualified fixed name for a server object. A fixed name does not change if the object is moved to another server. A qualified fixed name with a cell scope has the following form:

`cell/persistent/`*`fixedName`*

The *fixedName* is an arbitrary fixed name.

You can configure namespace bindings, and thus qualified fixed names, for the following objects:
- A string constant value
- An enterprise bean (EJB) home installed on a server in the cell
- A CORBA object available from a CosNaming name server
- An object bound in a WebSphere Application Server namespace that is accessible using a Java Naming and Directory Interface (JNDI) indirect lookup

To view or configure a namespace binding for an object of a deployed application, complete the following:

1. Go to the Name space bindings page.

   In the administrative console, click **Environment** → **Naming** → **Name space bindings**.
2. Select the desired scope.

   The scope determines where in the namespace binding is created. It also affects which name servers contain the binding in the namespace that they manage. Regardless of the scope, a namespace binding is accessible from all name servers in the cell. However, the scope can affect whether the lookup can be resolved locally by a name server or whether the name server must make a remote call to another name server to resolve the binding.

Only namespace bindings created with the selected scope are visible in the collection table on the page. By changing the scope, you can see and create bindings in other scopes.

a. Select a scope.

If you are creating a new namespace binding, refer to the table below as a guide in selecting a scope:

Scope	Description
Cell	Cell-scoped bindings are created under the cell persistent root context. Select **Cell** if the namespace binding is not specific to any particular node or server, or if you do not want the binding to be associated with any specific node or server. For example, you can use cell-scoped bindings to create fixed qualified names for enterprise beans. Fixed qualified names do not have any node or server names embedded within them.
Node	Node-scoped bindings are created under the node persistent root context for the selected node. Select **Node** if the namespace binding is specific to a particular node, or if you want the binding to be associated with a specific node.
Server	Server-scoped bindings are created under the server root context for the selected server. Select **Server** if a binding is to be used only by clients of an application running on a particular server, or if you want to configure a binding with the same name on different servers which resolve to different objects. Note that two servers can have configured bindings with the same name but resolve to different objects. Server-scoped bindings are created in the process of the selected application server. Therefore, the name server running in the selected application server can resolve those bindings locally. No remote invocations to other name servers are necessary to resolve the bindings. However, all other name servers in the cell must make remote calls to the selected server in order to resolve the bindings. For example, in order for the name server running in *server1* in node *node1* to resolve the name `cell/nodes/node1/servers/server2/serverScopedConfiguredBinding`, it must make a remote call to *server2* in *node1*. Only the name server in *server2* in *node1* can resolve that name without invoking any other name servers.

b. Click **Apply**.

3. Create a new namespace binding.

a. Open the New Name Space Binding wizard.

On the Name space bindings page, click **New**.

b. On the **Specify binding type** panel, select the binding type.

The namespace binding can be for a constant string value, an EJB home, a CORBA CosNaming NamingContext or CORBA leaf node object, or an object that you can look up indirectly using JNDI.

c. On the **Specify basic properties** panel, specify the binding identifier and other properties for the binding.

For property descriptions, refer to the following:
- “String binding settings” on page 2064
- “EJB binding settings” on page 2065
- “CORBA object binding settings” on page 2066
- “Indirect lookup binding settings” on page 2067

d. On the **Summary** panel, verify the settings and click **Finish**.

The name of the new binding is displayed in the collection table on the Name space bindings page.

4. Optional: Edit a previously created binding.

a. From the collection table on the Name space bindings page, click the name of the binding that you want to edit.

b. Edit the binding properties as desired. Step 3(c) provides links to property descriptions.

c. Click **OK**.

Results

Cell-scoped bindings are created under the cell persistent root context. Node-scoped bindings are created under the node persistent root context for the specified node. Server-scoped bindings are created under the server root context for the selected server.

Name space binding collection

Use this page to configure a name binding of an EJB, a CORBA CosNaming NamingContext, a CORBA leaf node object, an object that you can look up using Java Naming and Directory Interface (JNDI), or a constant string value.

Binding information for configured bindings is stored in the configuration and applied upon startup of the name server for each server within the scope of the binding.

To view the Name space bindings page, click **Environment** → **Naming** → **Name space bindings**.

Click the check boxes to select one or more of the bindings in your collection. Use the buttons to control the selected bindings.

When creating a new binding, select a scope before clicking **New**. The **Scope** setting filters what bindings are listed in the collection as well as sets the scope of the new binding.

Name

Shows the names given to uniquely identify these configured bindings.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file. This field is for information purposes only and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Valid values are String, EJB, CORBA, and Indirect. This field is for information purposes only and cannot be updated.

Specify binding type settings

Use this panel to select the type of namespace binding that you want.

To view this administrative console panel, click **Environment** → **Naming** → **Name space bindings** → **New**.

You can configure a namespace binding for any of the following objects:

- A string constant value
- An enterprise bean (EJB) home installed on a server in the cell
- A CORBA object available from a CosNaming name server
- An object bound in a WebSphere Application Server namespace that is accessible using a Java Naming and Directory Interface (JNDI) indirect lookup

On this panel, select a binding type and then click **Next**.

Binding type

Specifies the type of binding configured.

String	Select **String** to configure a namespace binding for a string constant value. To configure a String binding, you need the following information: • The string constant value • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context You can create a file that maps multiple variable names to values and specify the file name for the **String** value. By default, a name server performs variable substitution on the string value of a String namespace binding. Thus, by default, the com.ibm.websphere.naming.expandStringBindings property is set to `true` and a name server expands the value of String bindings. **Note:** Variable substitution can result in errors or unexpected changes to a string. For example, with variable substitution, a `$$` string is expanded as `$`. You can disable variable substitution and cause the name server to treat the **String** value as a literal or constant. Create a custom property with **Name** set to `com.ibm.websphere.naming.expandStringBindings` and **Value** set to `false`. You can define a custom property at the cell, node, server, or name server scope. Create the custom property on a console page for the appropriate scope: **Cell scope** Click **System administration** → **Cell** → **Custom properties** → **New**. **Node scope** Click **System administration** → **Nodes** → ***node_name*** → **Custom properties** → **New**. **Server scope** Click **Application servers** → ***server_name*** → **Administration** → **Custom properties** → **New**. **Name server scope** Click **Application servers** → ***server_name*** → **Administration** → **Server components** → **Name server** → **Custom properties** → **New**. All name servers within the specified custom property scope apply the setting. Settings at a narrower scope override settings at a wider scope. For example, on multiple-server products, settings at the node scope override settings at the cell scope. Select a custom property scope that is at least as wide as the namespace binding scope. Thus, to prevent variable expansion in a cell-scoped String namespace binding, define the custom property at the cell scope. If the custom property has a scope narrower than the namespace binding, only name servers within the scope prevent variable expansion in the String namespace binding. Name servers outside of the scope expand the variable reference and handle the reference differently.
EJB	Select **EJB** to configure a namespace binding for an EJB home installed on a server in the cell. Use a cell-scoped EJB binding to create a fixed qualified lookup name for an enterprise bean. A fixed qualified lookup name is not dependent on the cell topology. To configure an EJB home binding, you need the following information: • The JNDI name of the EJB server or server cluster where the enterprise bean is deployed • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context On stand-alone servers, do not configure an EJB binding that resolves to another server. The name server cannot read configuration data for other servers. That data is required to construct the binding.
CORBA	Select **CORBA** to configure a namespace binding for a Common Object Request Broker: Architecture and Specification (CORBA) object available from a Object Management Group (OMG) Interoperable Naming (CosNaming) name server. Identify a CORBA object bound into an INS compliant CosNaming server with a corbaname URL. The referenced object does not have to be available until the binding is actually referenced by an application. To configure a CORBA binding, you need the following information: • The corbaname URL of the CORBA object • An indicator if the bound object is a context or leaf node object (to set the correct CORBA binding type of context or object) • The target root context for the configured binding • The name of the configured binding, relative to the target root context

Indirect	Select **Indirect** to configure a namespace binding for an object bound in a WebSphere Application Server namespace that is accessible using a JNDI indirect lookup. You can select **Indirect** for CORBA objects as well as for javax.naming.Referenceable, javax.naming.Reference, and java.io.Serializable objects. The target object itself is not bound to the namespace. Only the information required to look up the object is bound. Therefore, the referenced name server does not have to be running until the binding is actually referenced by some application. To configure an indirect JNDI lookup binding, you need the following information: • The JNDI provider URL of the name server where the object resides • The JNDI lookup name of the object • The target root context for the configured binding (scope) • The name of the configured binding, relative to the target root context The following information is optional: • The JNDI initial context factory class name. The default is the WebSphere Application Server initial context factory, com.ibm.websphere.naming.WsnInitialContextFactory. • Additional properties to pass to the javax.naming.InitialContext constructor. A cell-scoped indirect binding is useful when creating a fixed qualified lookup name for a bound object so that the qualified lookup name is not dependent on the cell topology.

String binding settings

Use this page to view or configure a string binding.

To view this console page, click **Environment** → **Naming** → **Name space bindings** → ***string_namespace_binding***. The settings on this page are similar to those on the **Specify basic properties** panel of the New name space binding wizard.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier

Specifies the name that uniquely identifies this configured binding.

Name in name space

Specifies the name used for this binding in the namespace. This name can be a simple or compound name depending on the portion of the namespace where this binding is configured.

String value

Specifies the string to be bound into the namespace.

You can create a file that maps multiple variable names to values and specify the file name for the **String** value. By default, a name server performs variable substitution on the string value of a String namespace binding. Thus, by default, the com.ibm.websphere.naming.expandStringBindings property is set to `true` and a name server expands the value of String bindings.

Note: Variable substitution can result in errors or unexpected changes to a string. For example, with variable substitution, a `$$` string is expanded as `$`. You can disable variable substitution and cause the name server to treat the **String** value as a literal or constant. Create a custom property with **Name** set to `com.ibm.websphere.naming.expandStringBindings` and **Value** set to `false`. You can define a custom property at the cell, node, server, or name server scope. Create the custom property on a console page for the appropriate scope:

Cell scope
Click **System administration** → **Cell** → **Custom properties** → **New**.

Node scope
Click **System administration** → **Nodes** → ***node_name*** → **Custom properties** → **New**.

Server scope
Click **Application servers** → ***server_name*** → **Administration** → **Custom properties** → **New**.

Name server scope
Click **Application servers** → ***server_name*** → **Administration** → **Server components** → **Name server** → **Custom properties** → **New**.

All name servers within the specified custom property scope apply the setting. Settings at a narrower scope override settings at a wider scope. For example, on multiple-server products, settings at the node scope override settings at the cell scope. Select a custom property scope that is at least as wide as the namespace binding scope. Thus, to prevent variable expansion in a cell-scoped String namespace binding, define the custom property at the cell scope. If the custom property has a scope narrower than the namespace binding, only name servers within the scope prevent variable expansion in the String namespace binding. Name servers outside of the scope expand the variable reference and handle the reference differently.

EJB binding settings

Use this page to configure a new enterprise bean (EJB) binding, or to view or edit an existing EJB binding.

To view this console page, click **Environment** → **Naming** → **Name space bindings** → ***EJB_namespace_binding***. The settings on this page are similar to those on the **Specify basic properties** panel of the New name space binding wizard.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier

Specifies the name that uniquely identifies this configured binding.

Name in name space
Specifies the name used for this binding in the namespace. This name can be a simple or compound name depending on the portion of the namespace where this binding is configured.

Enterprise bean location
Specifies whether the enterprise bean is running in a single server. If `Single server` is specified, type the node name.

Server
Specifies the name of the server in which the enterprise bean is configured.

JNDI name
Specifies the Java Naming and Directory Interface (JNDI) name of the deployed enterprise bean. It is the bean's JNDI name that is in the enterprise bean bindings, not the java:comp name.

CORBA object binding settings
Use this page to configure a new name binding of a CORBA object binding, or to view or edit an existing CORBA object binding.

To view this console page, click **Environment** → **Naming** → **Name space bindings** → ***CORBA_namespace_binding***. The settings on this page are similar to those on the **Specify basic properties** panel of the New name space binding wizard.

Scope
Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file.

The **Scope** setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type
Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes and cannot be updated.

Binding identifier
Specifies the name that uniquely identifies this configured binding.

Name in name space
Specifies the name used for this binding in the namespace. This name can be a simple or compound name depending on the portion of the namespace where this binding is configured.

Corbaname URL
Specifies the CORBA name URL string identifying where the object is bound in a CosNaming server.

Federated context
Specifies whether the target is a CosNaming context (true) or a leaf node object (false).

true	The target object is bound with a context CORBA binding type. If the corbaname URL does not resolve to a NamingContext, an error occurs when the binding is first used (which is when the URL is first resolved).
false	The target object is bound with an object CORBA binding type.

Indirect lookup binding settings

Use this page to configure a new indirect lookup name binding, or to view or edit an existing indirect lookup binding.

To view this console page, click **Environment** → **Naming** → **Name space bindings** → ***indirect_lookup_namespace_binding***. The settings on this page are similar to those on the **Specify basic properties** panel of the New name space binding wizard.

Scope

Shows the scope of the configured binding. This value indicates the configuration location for the namebindings.xml file.

The **Scope** setting is shown only when you edit an existing binding on the console page, and is not shown when you create a new binding on the wizard panel. This setting is for information purposes and cannot be updated.

If the configured binding is cell-scoped, the starting context is the cell persistent root context. If the configured binding is node-scoped, the starting context is the node persistent root context. If the configured binding is server-scoped, the starting context is the server's server root context.

Binding type

Shows the type of binding configured. Possible choices are String, EJB, CORBA, and Indirect. This setting is for information purposes only and cannot be updated.

Binding identifier

Specifies the name that uniquely identifies this configured binding.

Name in name space

Specifies the name used for this binding in the namespace. This name can be a simple or compound name depending on the portion of the namespace where this binding is configured.

Provider URL

Specifies the provider URL string needed to obtain a Java Naming and Directory Interface (JNDI) initial context.

JNDI name

Specifies the name used to look up the target object from the initial context.

Initial context factory name

Specifies the class name of the initial context factory used to obtain a JNDI initial context.

This field is optional. If no factory is specified, the WebSphere Application Server initial context factory is used.

If the scope of the indirect lookup binding includes servers or nodes previous to Version 6.1, the initial context factory name and other context factory properties are not shown on the console page.

Developing applications that use JNDI

References to enterprise bean (EJB) homes and other artifacts such as data sources are bound to the WebSphere Application Server name space. These objects can be obtained through Java Naming and Directory Interface (JNDI). Before you can perform any JNDI operations, you need to get an initial context. You can use the initial context to look up objects bound to the name space.

About this task

The following examples describe how to get an initial context and how to perform lookup operations.
- Getting the default initial context
- Getting an initial context by setting the provider URL property
- Setting the provider URL property to select a different root context as the initial context
- Looking up an EJB home with JNDI

In these examples, the default behavior of features specific to the WebSphere Application Server JNDI Context implementation is used.

The WebSphere Application Server JNDI context implementation includes special features. JNDI caching enhances performance of repeated lookup operations on the same objects. Name syntax options offer a choice of a name syntaxes, one optimized for typical JNDI clients, and one optimized for interoperability with CosNaming applications. Most of the time, the default behavior of these features is the preferred behavior. However, sometimes you should modify the behavior for specific situations.

JNDI caching and name syntax options are associated with a javax.naming.InitialContext instance. To select options for these features, set properties that are recognized by the WebSphere Application Server initial context factory. To set JNDI caching or name syntax properties which will be visible to the initial context factory, do the following:

1. Optional: Configure JNDI caches

   JNDI caching can greatly increase performance of JNDI lookup operations. By default, JNDI caching is enabled. In most situations, this default is the desired behavior. However, in specific situations, use the other JNDI cache options.

   Objects are cached locally as they are looked up. Subsequent lookups on cached objects are resolved locally. However, cache contents can become stale. This situation is not usually a problem, since most objects you look up do not change frequently. If you need to look up objects which change relatively frequently, change your JNDI cache options.

   JNDI clients can use several properties to control cache behavior.

   You can set properties:

   - From the command line by entering the actual string value. For example:

     ```
     java -Dcom.ibm.websphere.naming.jndicache.maxentrylife=1440
     ```

   - In a jndi.properties file by creating a file named jndi.properties as a text file with the desired properties settings. For example:

     ```
     ...
     com.ibm.websphere.naming.jndicache.cacheobject=none
     ...
     ```

     Include the file as the beginning of the classpath, so that the class loader loads your copy of jndi.properties before any other copies.

   - Within a Java program by using the PROPS.JNDI_CACHE* Java constants, defined in the com.ibm.websphere.naming.PROPS file. The constant definitions follow:

     ```
     public static final String JNDI_CACHE_OBJECT =
      "com.ibm.websphere.naming.jndicache.cacheobject";
     public static final String JNDI_CACHE_OBJECT_NONE      = "none";
     public static final String JNDI_CACHE_OBJECT_POPULATED = "populated";
     public static final String JNDI_CACHE_OBJECT_CLEARED   = "cleared";
     public static final String JNDI_CACHE_OBJECT_DEFAULT   =
      JNDI_CACHE_OBJECT_POPULATED;

     public static final String JNDI_CACHE_NAME =
      "com.ibm.websphere.naming.jndicache.cachename";
     public static final String JNDI_CACHE_NAME_DEFAULT = "providerURL";

     public static final String JNDI_CACHE_MAX_LIFE =
      "com.ibm.websphere.naming.jndicache.maxcachelife";
     ```

```
public static final int    JNDI_CACHE_MAX_LIFE_DEFAULT = 0;

public static final String JNDI_CACHE_MAX_ENTRY_LIFE =
 "com.ibm.websphere.naming.jndicache.maxentrylife";
public static final int    JNDI_CACHE_MAX_ENTRY_LIFE_DEFAULT = 0;
```

To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the InitialContext constructor as follows:

```
java.util.Hashtable env = new java.util.Hashtable();
...

// Disable caching
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE); ...
javax.naming.Context initialContext = new javax.naming.InitialContext(env);
```

Example: Controlling JNDI cache behavior from a program

Following are examples that illustrate how you can use JNDI cache properties to achieve the desired cache behavior. Cache properties take effect when an InitialContext object is constructed.

```
import java.util.Hashtable;
import javax.naming.InitialContext;
import javax.naming.Context;
import com.ibm.websphere.naming.PROPS;

/*****
 Caching discussed in this section pertains to the WebSphere Application
 Server initial context factory. Assume the property,
 java.naming.factory.initial, is set to
 "com.ibm.websphere.naming.WsnInitialContextFactory" as a
 java.lang.System property.
*****/

Hashtable env;
Context ctx;

// To clear a cache:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_CLEARED);
ctx = new InitialContext(env);

// To set a cache's maximum cache lifetime to 60 minutes:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_MAX_LIFE, "60");
ctx = new InitialContext(env);

// To turn caching off:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
ctx = new InitialContext(env);

// To use caching and no caching:

env = new Hashtable();
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_POPULATED);
ctx = new InitialContext(env);
env.put(PROPS.JNDI_CACHE_OBJECT, PROPS.JNDI_CACHE_OBJECT_NONE);
Context noCacheCtx = new InitialContext(env);

Object o;

// Use caching to look up home, since the home should rarely change.
o = ctx.lookup("com/mycom/MyEJBHome");
// Narrow, etc. ...
```

```
// Do not use cache if data is volatile.
o = noCacheCtx.lookup("com/mycom/VolatileObject");
// ...
```

Example: Looking up a JavaMail session with JNDI

The following example shows a lookup of a JavaMail resource:

```
// Get the initial context as shown above
...
Session session =
     (Session) initialContext.lookup("java:comp/env/mail/MailSession");
```

2. Optional: Specify the name syntax

   INS syntax is designed for JNDI clients that need to interoperate with CORBA applications. This syntax allows a JNDI client to make the proper mapping to and from a CORBA name. INS syntax is very similar to the JNDI syntax with the additional special character, dot (.). Dots are used to delimit the id and kind fields in a name component. A dot is interpreted literally when it is escaped. Only one unescaped dot is allowed in a name component. A name component with a non-empty id field and empty kind field is represented with only the id field value and must not end with an unescaped dot. An empty name component (empty id and empty kind field) is represented with a single unescaped dot. An empty string is not a valid name component representation.

   JNDI name syntax is the default syntax and is suitable for typical JNDI clients. This syntax includes the following special characters: forward slash (/) and backslash (\). Components in a name are delimited by a forward slash. The backslash is used as the escape character. A forward slash is interpreted literally if it is escaped, that is, preceded by a backslash. Similarly, a backslash is interpreted literally if it is escaped.

   Most WebSphere applications use JNDI to look up EJB objects and do not need to look up objects bound by CORBA applications. Therefore, the default name syntax used for JNDI names is the most convenient. If your application needs to look up objects bound by CORBA applications, you may need to change your name syntax so that all CORBA CosNaming names can be represented.

   JNDI clients can set the name syntax by setting a property. The property setting is applied by the initial context factory when you instantiate a new java.naming.InitialContext object. Names specified in JNDI operations on the initial context are parsed according to the specified name syntax.

   You can set the property:

   - From a command line, enter the actual string value. For example:

     ```
     java -Dcom.ibm.websphere.naming.name.syntax=ins
     ```

   - Create a file named jndi.properties as a text file with the desired properties settings. For example:

     ```
     ...
     com.ibm.websphere.naming.name.syntax=ins
     ...
     ```

     Include the file at the beginning of the classpath, so that the class loader loads your copy of jndi.properties before any other copies.

   - Use the PROPS.NAME_SYNTAX* Java constants, defined in the com.ibm.websphere.naming.PROPS file, in a Java program. The constant definitions follow:

     ```
     public static final String NAME_SYNTAX =
         "com.ibm.websphere.naming.name.syntax";
     public static final String NAME_SYNTAX_JNDI = "jndi";
     public static final String NAME_SYNTAX_INS  = "ins";
     ```

     To use the previous properties in a Java program, add the property setting to a hashtable and pass it to the InitialContext constructor as follows:

     ```
     java.util.Hashtable env = new java.util.Hashtable();
     ...
     env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS); // Set name syntax to INS
     ...
     javax.naming.Context initialContext = new javax.naming.InitialContext(env);
     ```

   Example: Setting the syntax used to parse name strings

The name syntax property can be passed to the InitialContext constructor through its parameter, in the System properties, or in a jndi.properties file. The initial context and any contexts looked up from that initial context parse name strings based on the specified syntax.

The following example shows how to set the name syntax to make the initial context parse name strings according to INS syntax.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS; // WebSphere naming constants
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, ...);
env.put(PROPS.NAME_SYNTAX, PROPS.NAME_SYNTAX_INS);
Context initialContext = new InitialContext(env);
// The following name maps to a CORBA name component as follows:
//    id = "a.name", kind = "in.INS.format"
// The unescaped dot is used as the delimiter.
// Escaped dots are interpreted literally.
java.lang.Object o = initialContext.lookup("a\\.name.in\\.INS\\.format");
...
```

INS name syntax requires that embedded periods (.) in a name such as `in.INS.format` be escaped using a backslash character (\). In a Java String literal, a backslash character (\) must be escaped with another backslash character (\\).

Example: Getting the default initial context

There are various ways for a program to get the default initial context.

The following example gets the default initial context. Note that no provider URL is passed to the javax.naming.InitialContext constructor.

```
...
import javax.naming.Context;
import javax.naming.InitialContext;
...
Context initialContext = new InitialContext();
...
```

The default initial context returned depends the runtime environment of the Java Naming and Directory Interface (JNDI) client. Following are the initial contexts returned in the various environments:

Thin client
The initial context is the server root context of the server running on the local host at port 2809.

Pure client
The initial context is the context specified by the java.naming.provider.url property passed to launchClient command with the -CCD command line parameter. The context usually is the server root context of the server at the address specified in the URL, although it is possible to construct a corbaname or corbaloc URL which resolves to some other context.

If no provider URL is specified, it is the server root context of the server running on the host and port specified by the `-CCproviderURL`, or `-CCBootstrapHost` and `-CCBootstrapPort` command line parameters. The default host is the local host, and the default port is 2809.

Server process
The initial context is the server root context for that process.

Even though no provider URL is explicitly specified in the above example, the InitialContext constructor might find a provider URL defined in other places that it searches for property settings.

Users of properties which affect ORB initialization should read the rest of this section for a deeper understanding of exactly how initial contexts are obtained.

Determining which server is used to obtain the initial context

WebSphere Application Server name servers are CORBA CosNaming name servers, and the product provides a CosNaming JNDI plug-in implementation for JNDI clients to perform naming operations on product namespaces. The CosNaming plug-in implementation is selected through a JNDI property that is passed to the InitialContext constructor. This property is `java.naming.factory.initial`, and it specifies the initial context factory implementation to use to obtain an initial context. The factory returns a javax.naming.Context instance, which is part of its implementation.

The initial context factory, `com.ibm.websphere.naming.WsnInitialContextFactory`, is typically used by applications to perform JNDI operations. The WebSphere Application Server runtime environment is set up to use this initial context factory if one is not specified explicitly by the JNDI client. When the initial context factory is invoked, an *initial context* is obtained. The following paragraphs explain how the initial context factory obtains the initial context in client and server environments.

- **Registration of initial references in server processes**

  Every WebSphere Application Server has an ORB used to receive and dispatch invocations on objects running in that server. Services running in the server process can register initial references with the ORB. Each initial reference is registered under a key, which is a string value. An initial reference can be any CORBA object. WebSphere Application Server name servers register several initial contexts as initial references under predefined keys. Each name server initial reference is an instance of the interface org.omg.CosNaming.NamingContext.

- **Obtaining initial references in pure client processes**

  Pure JNDI clients, that is, JNDI clients which are not running in a WebSphere Application Server process, also have an ORB instance. This client ORB instance can be passed to the InitialContext constructor, but typically the initial context factory creates and initializes the client ORB instance transparently. A client ORB can be initialized with initial references, but the initial references most likely resolve to objects running in some server. The initial context factory does not define any default initial references when it initializes an ORB. If the `resolve_initial_references` method is invoked on the client ORB when no initial references have been configured, the method invocation fails. This condition is typical for pure client processes. To obtain an initial NamingContext reference, the initial context factory must invoke `string_to_object` with an IIOP type CORBA object URL, such as `corbaloc:iiop:myhost:2809`. The URL specifies the address of the server from which to obtain the initial context. The host and port information is extracted from the provider URL passed to the InitialContext constructor.

  If no provider URL is defined, the WebSphere Application Server initial context factory uses the default provider URL of `corbaloc:iiop:`*`your.server.name`*`:2809`.

  The `string_to_object` ORB method resolves the URL and communicates with the target server ORB to obtain the initial reference.

- **Obtaining initial references in server processes**

  If the JNDI client is running in a WebSphere Application Server process, the initial context factory obtains a reference to the server ORB instance if the JNDI client does not provide an ORB instance. Typically, JNDI clients running in server processes use the server ORB instance; that is, they do not pass an ORB instance to the InitialContext constructor. The name server which is running in the server process sets a provider URL as a java.lang.System property to serve as the default provider URL for all JNDI clients in the process. This default provider URL is `corbaloc:rir:/NameServiceServerRoot`. This URL resolves to the server root context for that server. (The URL is equivalent to invoking `resolve_initial_references` on the ORB with a key of NameServiceServerRoot. The name server registers the server root context as an initial reference under that key.)

- **Understanding the legacy ORB protocol**

Releases previous to WebSphere Application Server Version 5 used a different ORB implementation, which used a legacy protocol in contrast with the Interoperable Name Service (INS) protocol now used. This change has affected the implementation of the initial context factory. **Certain types of pure clients can experience different behavior when getting initial JNDI contexts as compared to previous releases of WebSphere Application Server**. This behavior is discussed in more detail below.

The following ORB properties are used with the legacy ORB protocol for ORB initialization and are now deprecated:
– com.ibm.CORBA.BootstrapHost
– com.ibm.CORBA.BootstrapPort

The new INS ORB is different in a major respect, in that it exhibits no default behavior if no initial references are defined.

In the legacy ORB, the bootstrap host and port values defaulted to *your.server.name* and 900.

All initial references were obtained from the server running on the bootstrap host and port. So, if the ORB user provided no bootstrap host and port, all initial references are resolved from the server running on the local host at port 900. The INS ORB has no concept of bootstrap host or bootstrap port. All initial references are defined independently. That is, different initial references could resolve to different servers. If ORB.resolve_initial_references is invoked with a key such that the ORB is not initialized with an initial reference having that key, the call fails.

In releases previous to Version 5, the initial context factory invoked resolve_initial_references on the ORB in the absence of any provider URL. This action succeeded if a name server at the default bootstrap host and port was running. In the current release, with the INS ORB, this would fail. (Actually, the ORB would fall back to the legacy protocol during the deprecation period, but when the legacy protocol is no longer supported, the operation would fail.)

The initial context factory now uses a default provider URL of corbaloc:iiop:*your.server.name*:2809, and invokes string_to_object with the provider URL.

This operation preserves the behavior that pure clients in previous releases experienced when they set no ORB bootstrap properties or provider URL. **However, this different initial context factory implementation changes the behavior experienced by certain legacy pure clients, which do not specify a provider URL**:
– Clients which set the ORB bootstrap properties listed above when getting an initial context.
– Clients which supply their own ORB instance to the InitialContext constructor.

There are two ways to circumvent this change of behavior:

– Always specify an IIOP type provider URL. This approach does not depend on the bootstrap host and port properties and continues to work when support for the bootstrap host and port properties is removed. For example, you can express bootstrap host and port property values of myHost and 2809, respectively, as corbaloc:iiop:myHost:2809.
– Use an rir type provider URL:
  - Specify corbaloc:rir:/NameServiceServerRoot if the ORB is initialized to use a Version 5 server as the bootstrap server.
  - Specify corbaname:rir:/NameService#domain/legacyRoot if the ORB is initialized to use a Version 4.0.x server as the bootstrap server.
  - Specify corbaloc:rir:/NameService if the ORB is initialized to use a server other than a Version 5 or 4.0.x server as the bootstrap server.

  URLs of this type are equivalent to invoking resolve_initial_references on the ORB with the specified key. If the bootstrap host and port properties are being used to initialize the ORB, this approach will not work when the bootstrap and host properties are no longer supported.

• **The InitialContext constructor search order for JNDI properties**

  If the code snippet shown at the beginning of this section is executed by an application, the bootstrap server depends on the value of the property, java.naming.provider.url.

  If the property is not set (in server processes the default value is set as a system property), the default host of *your.server.name* and default port of 2809 are used as the address of the server from which to obtain the initial context.

The JNDI specification describes where the InitialContext constructor looks for java.naming.provider.url property settings, but briefly, the property is picked up from the following places in the order shown:

InitialContext constructor
This does not apply to the above example because the example uses the empty InitalContext constructor.

System environment
You can add JNDI properties to the system environment as an option on the Java command invocation and by program code. The recommended way to set the provider URL in the system environment is as an option supplied to the Java command invocation. Setting the provider URL in this manner is not temporal, so that getting a default initial context will always yield the same result. It is generally recommended that program code not set the provider URL property in the system environment because as a side-effect, this could adversely affect other, possibly unrelated, code running elsewhere in the same process.

jndi.properties file
There may be many jndi.properties files that are within the scope of the class loader in effect. All jndi.properties files are used for setting JNDI properties, but the provider URL setting is determined by the first jndi.properties file returned by the class loader.

Example: Getting an initial context by setting the provider URL property

In general, Java Naming and Directory Interface (JNDI) clients should assume the correct environment is already configured so there is no need to explicitly set property values and pass them to the `InitialContext` constructor. However, a JNDI client might need to access a namespace other than the one identified in its environment. In this case, it is necessary to explicitly set the java.naming.provider.url (provider URL) property used by the `InitialContext` constructor. A provider URL contains bootstrap server information that the initial context factory can use to obtain an initial context. Any property values passed in directly to the `InitialContext` constructor take precedence over settings of those same properties found elsewhere in the environment.

You can use two different provider URL forms with theWebSphere Application Server initial context factory:
- A CORBA object URL
- An IIOP URL

CORBA object URLs are more flexible than IIOP URLs and are the recommended URL format to use. CORBA object URLs are part of the OMG CosNaming Interoperable Naming Specification. A corbaname URL, for example, can include initial context and lookup name information and can be used as a lookup name without the need to explicitly obtain another initial context. The IIOP URLs are the legacy JNDI format, but are still supported by the WebSphere Application Server initial context factory.

The following examples illustrate the use of these URLs.

- “Using a CORBA object URL”
- “Using a CORBA object URL from a non-WebSphere Application Server JNDI implementation” on page 2075
- “Using an IIOP URL” on page 2075

Using a CORBA object URL

This example shows a CORBA object URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
```

```
     "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...
```

Using a CORBA object URL from a non-WebSphere Application Server JNDI implementation

Initial context factories for CosNaming JNDI plug-in implementations other than the WebSphere Application Server initial context factory most likely obtain an initial context using the object key, `NameService`. When you use such a context factory to obtain an initial context from a WebSphere Application Server name server, the initial context is the cell root context. Since system artifacts such as EJB homes associated with a server are bound under the server's server root context, names used in JNDI operations must be qualified. If you want to use relative names, ensure your initial context is the server root context under which the target object is bound. In order to make the server root context the initial context, specify a corbaloc provider URL with an object key of `NameServiceServerRoot`.

This example shows a CORBA object type URL from a non-WebSphere Application Server JNDI implementation. This example assumes full CORBA object URL support by the non-WebSphere Application Server JNDI implementation. The object key of `NameServiceServerRoot` is specified so that the initial context will be the specified server's server root context.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.somecompany.naming.TheirInitialContextFactory");
env.put(Context.PROVIDER_URL,
      "corbaname:iiop:myhost.mycompany.com:9810/NameServiceServerRoot");
Context initialContext = new InitialContext(env);
...
```

If qualified names are used, you can use the default key of `NameService`.

Using an IIOP URL

The IIOP type of URL is a legacy format which is not as flexible as CORBA object URLs. However, URLs of this type are still supported. The following example shows an IIOP type URL as the provider URL.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "iiop://myhost.mycompany.com:2809");
Context initialContext = new InitialContext(env);
...
```

Example: Setting the provider URL property to select a different root context as the initial context

Each server contains its own server root context, and, when bootstrapping to a server, the server root is the default initial JNDI context. Most of the time, this default is the desired initial context, since system artifacts such as EJB homes are bound there. However, other root contexts exist, which can contain bindings of interest. It is possible to specify a provider URL to select other root contexts.

Examples for selecting other root contexts follow:

- Initial root contexts with a CORBA object URL

- Initial root contexts with the namespace root property

Selecting the initial root context with a CORBA object URL

There are several object keys registered with the bootstrap server that you can use to select the root context for the initial context. To select a particular root context with a CORBA object URL object key, set the object key to the corresponding value. The default object key is NameService. Using JNDI yields the server root context. A table that lists the different root contexts and their corresponding object key follows:

Root Context	CORBA Object URL Object Key
Server Root	NameServiceServerRoot
Cell Persistent Root	NameServiceCellPersistentRoot
Cell Root	NameServiceCellRoot
Node Root	NameServiceNodeRoot

The following example shows the use of a corbaloc URL with the object key set to select the cell persistent root context as the initial context.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
...
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
      "corbaloc:iiop:myhost.mycompany.com:2809/NameServiceCellPersistentRoot");
Context initialContext = new InitialContext(env);
...
```

Selecting the initial root context with the namespace root property

You can also select the initial root context by passing a namespace root property setting to the InitialContext constructor. Generally, the object key setting described above is sufficient. Sometimes a property setting is preferable. For example, you can set the root context property on the Java invocation to make which server root is being used as the initial context transparent to the application. The default server root property setting is `defaultroot`, which yields the server root context.

Root Context	Namespace Root Property Value
Server Root	bootstrapserverroot
Cell Persistent Root	cellpersistentroot
Cell Root	cellroot
Node Root	bootstrapnoderoot

The initial context factory ignores the namespace root property if the provider URL contains an object key other than `NameService`.

The following example shows use of the namespace root property to select the cell persistent root context as the initial context. Note that available constants are used instead of hard-coding the property name and value.

```
...
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import com.ibm.websphere.naming.PROPS;
...
Hashtable env = new Hashtable();
```

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
      "com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL, "corbaloc:iiop:myhost.mycompany.com:2809");
env.put(PROPS.NAME_SPACE_ROOT, PROPS.NAME_SPACE_ROOT_CELL_PERSISTENT);
Context initialContext = new InitialContext(env);
...
```

Example: Looking up an EJB home or business interface with JNDI

Most applications that use Java Naming and Directory Interface (JNDI) run in a container. Some do not. The name used to look up an object depends on whether or not the application is running in a container. Sometimes it is more convenient for an application to use a corbaname URL as the lookup name. Container-based JNDI clients and thin Java clients can use a corbaname URL.

The following examples show how to perform JNDI lookups from different types of applications.

- "JNDI lookup from an application running in a container"
- "JNDI lookup from an application that does not run in a container" on page 2078
- "JNDI lookup with a corbaname URL" on page 2079

JNDI lookup from an application running in a container

Applications that run in a container can use `java:` lookup names. Lookup names of this form provide a level of indirection such that the lookup name used to look up an object is not dependent on the object's name as it is bound in the name server's namespace. The deployment descriptors for the application provide the mapping from the `java:` name and the name server lookup name. The container sets up the `java:` namespace based on the deployment descriptor information so that the `java:` name is correctly mapped to the corresponding object.

The following example shows a lookup of an EJB 3.0 remote business interface. The actual home lookup name is determined by the interface's ibm-ejb-jar-bnd.xml binding file, if present, or by the default name assigned by the EJB container if no binding file is present. For more information, see Default bindings for business interfaces and homes and User-defined bindings for EJB business interfaces and homes.

```
// Get the initial context as shown in a previous example.
...
// Look up the business interface using the JNDI name.
try {
   java.lang.Object ejbBusIntf =
      initialContext.lookup(
         "java:comp/env/com/mycompany/accounting/Account");
   accountIntf =
      (Account)javax.rmi.PortableRemoteObject.narrow(ejbBusIntf, Account.class);
}
   catch (NamingException e) { // Error getting the business interface
   ...
}
```

The following example shows a lookup of an EJB 1.x or 2.x EJB home. The actual home lookup name is determined by the application's deployment descriptors. The enterprise bean (EJB) resides in an EJB container, which provides an interface between the bean and the application server on which it resides.

```
// Get the initial context as shown in a previous example
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome =
      initialContext.lookup(
         "java:comp/env/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
```

```
}
   catch (NamingException e) { // Error getting the home interface
   ...
}
```

JNDI lookup from an application that does not run in a container

Applications that do not run in a container cannot use `java:` lookup names because it is the container which sets the `java:` namespace up for the application. Instead, an application of this type must look the object up directly from the name server. Each application server contains a name server. System artifacts such as EJB homes are bound relative to the server root context in that name server. The various name servers are federated by means of a system namespace structure. The recommended way to look up objects on different servers is to qualify the name so that the name resolves from any initial context in the cell. If a relative name is used, the initial context must be the same server root context as the one under which the object is bound. The form of the qualified name depends on whether the qualified name is a topology-based name or a fixed name. Examples of each form of qualified name follow.

- **Topology-based qualified names**

  Topology-based qualified names traverse through the system namespace to the server root context under which the target object is bound. A topology-based qualified name resolves from any initial context in the cell.

  Single server

  The following example shows a lookup of an EJB business interface that is running in the single server, MyServer, configured in the node, Node1.

```
// Get the initial context as shown in a previous example.
// Using the form of lookup name below, it does not matter which
// server in the cell is used to obtain the initial context.
...
// Look up the business interface using the JNDI name
try {
   java.lang.Object ejbBusIntf = initialContext.lookup(
      "cell/nodes/Node1/servers/MyServer/com/mycompany/accounting/Account");
   accountIntf =
     (Account)javax.rmi.PortableRemoteObject.narrow(ejbBusIntf, Account.class);
}
catch (NamingException e) { // Error getting the business interface
   ...
}
```

  The following example shows a lookup of an EJB home that is running in the single server, MyServer, configured in the node, Node1.

```
// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome = initialContext.lookup(
      "cell/nodes/Node1/servers/MyServer/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
}
catch (NamingException e) { // Error getting the home interface
   ...
}
```

- **Fixed qualified names**

  If the target object has a cell-scoped fixed name defined for it, you can use its qualified form instead of the topology-based qualified name. Even though the topology-based name works, the fixed name does not change with the specific cell topology or with the movement of the target object to a different server.

  An example lookup of an EJB business interface with a qualified fixed name follows.

```
// Get the initial context as shown in a previous example.
// Using the form of lookup name below, it does not matter which
// server in the cell is used to obtain the initial context.
...
// Look up the business interface using the JNDI name
try {
   java.lang.Object ejbBusIntf = initialContext.lookup(
     "cell/persistent/com/mycompany/accounting/Account");
   accountIntf =
     (Account)javax.rmi.PortableRemoteObject.narrow(ejbBusIntf, Account.class);
   }
catch (NamingException e) { // Error getting the business interface
...

}
```

An example lookup with a qualified fixed name is shown below.

```
// Get the initial context as shown in a previous example
// Using the form of lookup name below, it doesn't matter which
// server in the cell is used to obtain the initial context.
...
// Look up the home interface using the JNDI name
try {
   java.lang.Object ejbHome = initialContext.lookup(
     "cell/persistent/com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
   }
catch (NamingException e) { // Error getting the home interface
...
}
```

JNDI lookup with a corbaname URL

A corbaname can be useful at times as a lookup name. If, for example, the target object is not a member of the federated namespace and cannot be located with a qualifiied name, a corbaname can be a convenient way to look up the object.

A lookup of an EJB business interface with a corbaname URL follows.

```
// Get the initial context as shown in a previous example.
...
// Look up the business interface using a corbaname URL.
try {
   java.lang.Object ejbBusIntf = initialContext.lookup(
      "corbaname:iiop:someHost:2809#com/mycompany/accounting/Account");
   accountIntf =
     (Account)javax.rmi.PortableRemoteObject.narrow(ejbBusIntf, Account.class);
}
catch (NamingException e) { // Error getting the business interface
   ...
}
```

A lookup with a corbaname URL follows.

```
// Get the initial context as shown in a previous example
...
// Look up the home interface using a corbaname URL
try {
   java.lang.Object ejbHome = initialContext.lookup(
      "corbaname:iiop:someHost:2809#com/mycompany/accounting/AccountEJB");
   accountHome = (AccountHome)javax.rmi.PortableRemoteObject.narrow(
      (org.omg.CORBA.Object) ejbHome, AccountHome.class);
```

```
}
catch (NamingException e) { // Error getting the home interface
   ...
}
```

JNDI interoperability considerations

You must take extra steps to enable your programs to interoperate with non-product JNDI clients and to bind resources from MQSeries to a namespace.

EJB clients running in an environment other than WebSphere Application Server accessing EJB applications running on WebSphere Application Server servers

When an enterprise bean (EJB) application running in WebSphere Application Server is accessed by a non-product EJB client, the JNDI initial context factory is presumed to be a non-product implementation. In this case, the default initial context is the cell root. If the JNDI service provider being used supports CORBA object URLs, the corbaname format can be used to look up the EJB home.

Single server

Following is a URL that has the bootstrap host **myHost**, the port **2809**, and the enterprise bean installed in the server **server1** in node **node1** and bound in that server under the name **myEJB**:

```
initialContext.lookup(
  "corbaname:iiop:myHost:2809#cell/nodes/node1/servers/server1/myEJB");
```

Without CORBA object URL support

If the JNDI initial context factory being used does not support CORBA object URLs, the initial context can be obtained from the server, and the lookup can be performed on the initial context as follows:

```
Hashtable env = new Hashtable();
env.put(CONTEXT.PROVIDER_URL, "iiop://myHost:2809");
Context ic = new InitialContext(env);
Object o = ic.lookup("cell/clusters/myCluster/myEJB");
```

Binding resources from MQSeries 5.2

In releases previous to WebSphere Application Server Version 5.0, the MQSeries jmsadmin tool could be used to bind resources to the namespace. When used with a WebSphere Application Server namespace, the resource is bound within a transient partition in the namespace and does not persist past the life of the server process. Instead of binding the MQSeries resources with the jmsadmin tool, bind them from the administrative console, under **Resources** in the console navigation tree.

JNDI caching

To increase the performance of Java Naming and Directory Interface (JNDI) operations, the product JNDI implementation employs caching to reduce the number of remote calls to the name server for lookup operations. For most cases, use the default cache setting.

When an InitialContext object is instantiated, an association is established between the InitialContext instance and a cache. The initial context and any contexts returned directly or indirectly from a lookup on the initial context are all associated with that same cache instance. By default, the association is based on the provider URL, in particular, the host name and port. The caller can specify the cache name to override this default behavior. A cache instance of a given name is shared by all instances of InitialContext configured to use a cache of that name which were created with the same context class loader in effect. Two enterprise bean (EJB) applications running in the same server will use their own cache instances, if they are using different context class loaders, even if the cache names are the same.

After an association between an InitialContext instance and cache is established, the association does not change. A `javax.naming.Context` object returned from a lookup operation inherits the cache association of the Context object on which the lookup was performed. Changing cache property values with the

`Context.addToEnvironment()` or `Context.removeFromEnvironment()` method does not affect cache behavior. You can change properties affecting a given cache instance with each InitialContext instantiation.

A cache is restricted to a process and does not persist past the life of that process. A cached object is returned from lookup operations until either the maximum cache life for the cache is reached, or the maximum entry life for the object's cache entry is reached. After this time, a lookup on the object causes the cache entry for the object to be refreshed. By default, caches and cache entries have unlimited lifetimes.

Usually, cached objects are relatively static entities, and objects becoming stale is not a problem. However, you can set timeout values on cache entries or on a cache so that cache contents are periodically refreshed.

If a bind or rebind operation is executed on an object, the change is not reflected in any caches other than the one associated with the context from which the bind or rebind was issued. This scenario is most likely to happen when multiple processes are involved, since different processes do not share the same cache, and context objects in all threads in a process typically share the same cache instance for a given name service provider.

JNDI cache settings

Various Java Naming and Directory Interface (JNDI) cache property settings follow. Ensure that all property values are string values.

com.ibm.websphere.naming.jndicache.cachename

The name of the cache to associate with an initial context instance can be specified with this property.

It is possible to create multiple InitialContext instances, each operating on the namespace of a different name server. By default, objects from each bootstrap address are cached separately, since they each involve independent namespaces and name collisions could occur if they used the same cache. The provider URL specified when the initial context is created by default serves as the basis for the cache name. With this property, a JNDI client can specify a cache name. Valid options for cache names follow:

Valid options	Resulting cache behavior
providerURL (default)	Use the value for java.naming.provider.url property as the basis for the cache name. Cache names are based on the bootstrap host and port specified in the URL. The boostrap host is normalized to a fully qualfied name, if possible. For example, ″corbaname:iiop:server1:2809#some/starting/context″ and ″corbaloc:iiop://server1″ are normalized to the same cache name. If no provider URL is specified, a default cache name is used.
Any string	Use the specified string as the cache name. You can use any arbitrary string with a value other than ″providerURL″ as a cache name.

com.ibm.websphere.naming.jndicache.cacheobject

Turn caching on or off and clear an existing cache with this property.

By default, when an InitialContext is instantiated, it is associated with an existing cache or, if one does not exist, a new one is created. An existing cache is used with its existing contents. In some circumstances, this behavior is not desirable. For example, when objects that are looked up change frequently, they can become stale in the cache. Other options are available. The following table lists these other options along with the corresponding property value.

Valid values	Resulting cache behavior
populated (default)	Use a cache with the specified name. If the cache already exists, leave existing cache entries in the cache; otherwise, create a new cache.

cleared	Use a cache with the specified name. If the cache already exists, clear all cache entries from the cache; otherwise, create a new cache.
none	Do not cache. If this option is specified, the cache name is irrelevant. Therefore, this option will not disable a cache that is already associated with other InitialContext instances. The InitialContext that is instantiated is not associated with any cache.

com.ibm.websphere.naming.jndicache.maxcachelife

Impose a limit to the age of a cache with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the com.ibm.websphere.naming.jndicache.cacheobject property set to `cleared`. This property enables a JNDI client to set the maximum life of a cache. This property differs from the maxentrylife property (below) in that the entire cache is cleared when the cache lifetime is reached. The table below lists the various maxcachelife values and their affect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Make the cache lifetime unlimited.
Positive integer	Set the maximum lifetime of the entire cache, in minutes, to the specified value. When the maximum lifetime for the cache is reached, the next attempt to read any entry from the cache causes the cache to be cleared

com.ibm.websphere.naming.jndicache.maxentrylife

Impose a limit to the age of individual cache entries with this property.

By default, cached objects remain in the cache for the life of the process or until cleared with the com.ibm.websphere.naming.jndicache.cacheobject property set to `cleared`. This property enables a JNDI client to set the maximum lifetime of individual cache entries. This property differs from the maxcachelife property in that individual entries are refreshed individually as their maximum lifetime reached. This might avoid any noticeable change in performance that might occur if the whole cache is cleared at once. The table below lists the various maxentrylife values and their effect on cache behavior:

Valid options	Resulting cache behavior
0 (default)	Lifetime of cache entries is unlimited.
Positive integer	Set the maximum lifetime of individual cache entries, in minutes, to the specified value. When the maximum lifetime for an entry is reached, the next attempt to read the entry from the cache causes the individual cache entry to refresh.

JNDI to CORBA name mapping considerations

WebSphere Application Server name servers are an implementation of the CORBA CosNaming interface. The product provides a Java Naming and Directory Interface (JNDI) implementation which you can use to access CosNaming name servers through the JNDI interface. Issues can exist when mapping JNDI name strings to and from CORBA names.

Each component in a CORBA name consists of an `id` and `kind` field, but a JNDI name component consists of no such fields. Each component in a JNDI name is atomic. Typical JNDI clients do not need to make a distinction between the `id` and `kind` fields of a name component, or know how JNDI name strings map to CORBA names. JNDI clients of this sort can use the JNDI syntax described below. When a name is parsed according to JNDI syntax, each name component is mapped to the `id` field of the corresponding CORBA name component. The `kind` field always has an empty value. This basic syntax is the least obtrusive to the JNDI client in that it has the fewest special characters. However, you cannot represent with this syntax a CORBA name with a non-empty `kind` field. This restriction can prevent EJB applications from interoperating with CORBA applications.

Some clients, however must interoperate with CORBA applications which use CORBA names with non-empty `kind` fields. These JNDI clients must make a distinction between `id` and `kind` so that JNDI names are correctly mapped to CORBA names, particularly when the CORBA names contain components with non-empty `kind` fields. Such JNDI clients can use the INS name syntax. With its additional special character, you can use INS to represent any CORBA name. Use of this syntax is not recommended unless it is necessary, because this syntax is more restrictive from the JNDI client's perspective in that the JNDI client must be aware that name components with multiple unescaped dots are syntactically invalid. INS name syntax is part of the OMG CosNaming Interoperable Naming Specification.

Chapter 12. Object Request Broker

Managing Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers using the Internet InterORB Protocol (IIOP). There are several ways to manage an ORB. For example, you can use ORB custom property settings, or system property settings to configure an ORB, or you can provide objects during ORB initialization.

About this task

Default ORB property values are set when the product starts and the ORB service initializes. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. You might have to modify some ORB settings to fit your system requirements.

After an ORB instance is established in a process, changes to ORB properties do not affect the behavior of a running ORB instance. You must stop the process and restart it before the modified settings take effect.

A list of possible tasks for managing an ORB follows.

- Adjust timeout settings to improve handling of network failures. Before making these adjustments, review the Object Request Broker tuning guidelines.
- Tune the ORB. For example, if most of your initial method invocations are very small, you might want to set the com.ibm.CORBA.enableLocateRequest custom property to false.
- Adjust the size of General Inter-ORB Protocol (GIOP) fragments that the ORB uses. You might want to make this adjustment if your applications frequently send large requests.
- Change the port that the ORB listens on.
- Specify an alternative to the default RAS manager of the ORB.
- Change the maximum number of connection requests that can remain unhandled by the product ORB before the application server starts to reject new incoming connection requests.
- Adjust thread-pool settings used by the ORB for handling Internet InterORB Protocol (IIOP) connections.
- Troubleshoot an ORB problem.

  If you experience problems with the ORB, you should review the Object request broker troubleshooting tips. If necessary, you can then enable ORB tracing, and then review the contents of the ORB communications trace.
- Adjust the logical pool distribution mechanism settings.

Object Request Brokers

An Object Request Broker (ORB) manages the interaction between clients and servers, using the Internet InterORB Protocol (IIOP). It enables clients to make requests and receive responses from servers in a network-distributed environment.

The ORB provides a framework for clients to locate objects in the network and to call operations on those objects as if the remote objects are located in the same running process as the client, providing location transparency. The client calls an operation on a local object, known as a *stub*. The stub forwards the request to the remote object, where the operation runs and the results are returned to the client.

The client-side ORB is responsible for creating an IIOP request that contains the operation and required parameters, and for sending the request on the network. The server-side ORB receives the IIOP request, locates the target object, invokes the requested operation, and returns the results to the client. The

client-side ORB demarshals the returned results and passes the result to the stub, which, in turn, returns to the client application, as if the operation had been run locally.

This product uses an ORB to manage communication between client applications and server applications as well as communication among product components. During product installation, default property values are set when the ORB is initialized. These properties control the run-time behavior of the ORB and can also affect the behavior of product components that are tightly integrated with the ORB, such as security. This product does not support the use of multiple ORB instances.

Logical pool distribution

The Logical pool distribution (LPD) thread pool mechanism implements a strategy for improving the performance of requests that have shorter run times. Do not configure LPD unless you have already configured it in a previous release of the product.

Note: LPD is a deprecated function and will be removed in a future version of the product.

The need for LPD is indicated by a mixture of Enterprise JavaBeans (EJB) requests where the run times vary across the request types, and the ORB thread pool must be constrained for performance reasons. In this case, longer run time requests might tend to prolong the response times for shorter requests by denying them adequate access to threads in the thread pool. LPD provides a mechanism that allows shorter requests greater access to the threads.

LPD divides the Object Request Broker (ORB) thread pool into logical pools, as configured by the administrator using ORB custom properties starting that start with the following:

```
com.ibm.websphere.threadpool.strategy.*
```

The size of each pool is a percentage of the maximum number of ORB threads. The sum of the logical pool percentages must equal 100.

When LPD is active, incoming ORB requests are vectored, or pointed, to a pool based on historical run time history for the request type. The request type is determined by the method, which is qualified internally as unique across components. The LPD mechanism adjusts pool targets at runtime to optimize the distribution of requests across logical pools.

The LPD mechanism can be tuned after it is enabled. Response time, throughput measurements, and statistics produced by the LPD mechanism drive the tuning process.

Object Request Broker tuning guidelines

Use the guidelines in this document any time the Object Request Broker (ORB) is used in a workload.

The ORB is used whenever enterprise beans are accessed through a remote interface. If you experience particularly high or low CPU consumption, you might have a problem with the value of one of the following parameters. Examine these core tuning parameters for every application deployment.

Thread pool adjustments

Size

Tune the size of the ORB thread pool according to your workload. Avoid suspending threads because they have no work ready to process. If threads do not have work ready to process, CPU time is consumed by calling the Object.wait method, performing a context switch. Tune the thread pool size such that the length of time that the threads wait is short enough to prevent them from being destroyed because they are idle too long.

The thread pool size is dependent on your workload and system. In typical configurations, applications need 10 or fewer threads per processor.

However, if your application is performing a very slow backend request, like a request to a database system, a server thread blocks waiting for the backend request to complete. With backend requests, CPU use is fairly low. In this case, increasing the load does not increase CPU use or throughput. Your thread dumps indicate that nearly all the threads are in a call out to the backend resource. In this case, consider increasing the number of threads per processor until throughput improves and thread dumps show that the threads are in other areas of the run time besides the backend call. You should adjust the number of threads only if your backend resource is tuned correctly.

The **Allow thread allocation beyond maximum thread size** parameter also affects thread pool size, but do not use this parameter unless your back end stops for long periods of time, causing the blocking of all the run-time threads waiting for the backend system instead of processing other work that does not involve the backend system.

You can adjust the thread pool size settings in the administrative console. Click **Servers > Server Types > Application servers >** *server_name* **> Container services > ORB service > Thread pool**. You can adjust the minimum and maximum number of threads.

Fragment size

The ORB separates messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.

To determine and change the size of the messages that transfer over the ORB and the number of required fragments, perform the following steps:

1. In the administrative console, enable ORB tracing in the ORB Properties page.
2. Enable ORBRas tracing from the logging and tracing page.
3. Increase the trace file sizes because tracing can generate a lot of data.
4. Restart the server and run at least one iteration (preferably several) of the case that you are measuring.
5. Look at the traceable file and do a search for Fragment to follow: Yes.

   This message indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send before the entire message arrives. A Fragment to follow: No value indicates that the particular fragment is the last in the entire message. This fragment can also be the first, if the message fit entirely into one fragment.

   If you go to the spot where Fragment to follow: Yes is located, you find a block that looks similar to the following example:

   Fragment to follow: Yes
   Message size: 4988 (0x137C)
   --
   Request ID: 1411

   This example indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. If you search for all occurrences of Request ID: 1411, you can see the number of fragments that are used to send that particular message. If you add all the associated message sizes, you have the total size of the message that is being sent through the ORB.
6. You can configure the fragment size by setting the com.ibm.CORBA.FragmentSize ORB custom property.

Interceptors

Interceptors are ORB extensions that can set up the context before the ORB runs a request. For example, the context might include transactions or activity sessions to import. If the client creates a transaction, and then flows the transaction context to the server, then the server imports the transaction context onto the server request through the interceptors.

Most clients do not start transactions or activity sessions, so most systems can benefit from removing the interceptors that are not required.

To remove the interceptors, manually edit the server.xml file and remove the interceptor lines that are not needed from the ORB section.

Connection Cache Adjustments

Depending on an application server's workload, and throughput or response-time requirements, you might need to adjust the size of the ORB's connection cache. Each entry in the connection cache is an object that represents a distinct TCP/IP socket endpoint, identified by the hostname or TCP/IP address, and the port number used by the ORB to send a GIOP request or a GIOP reply to the remote target endpoint. The purpose of the connection cache is to minimize the time required to establish a connection by reusing ORB connection objects for subsequent requests or replies. (The same TCP/IP socket is used for the request and corresponding reply.)

For each application server, the number of entries in the connection cache relates directly to the number of concurrent ORB connections. These connections consist of both the inbound requests made from remote clients and outbound requests made by the application server. When the server-side ORB receives a connection request, it uses an existing connection from an entry in the cache, or establishes a new connection and adds an entry for that connection to the cache.

The ORB Connection cache maximum and Connection cache minimum properties are used to control the maximum and minimum number of entries in the connection cache at a given time. When the number of entries reaches the value specified for the Connection cache maximum property, and a new connection is needed, the ORB creates the requested connection, adds an entry to the cache and searches for and attempts to remove up to five inactive connection entries from the cache. Because the new connection is added before inactive entries are removed, it is possible for the number of cache entries to temporarily exceed the value specified for the Connection cache maximum property.

An ORB connection is considered inactive if the TCP/IP socket stream is not in use and there are no GIOP replies pending for any requests made on that connection. As the application workload diminishes, the ORB closes the connections and removes the entries for these connections from the cache. The ORB continues to remove entries from the cache until the number of remaining entries is at or below the value specified for the Connection cache maximum property. The number of cache entries is never less then the value specified for the Connection cache minimum property, which must be at least five connections less than the value specified for the Connection cache maximum property.

Adjustments to the connection cache in the client-side ORB are usually not necessary because only a small number of connections are made on that side.

JNI Reader Threads

By default, the ORB uses a Java thread for processing each inbound connection request it receives. As the number of concurrent requests increases, the storage consumed by a large number of reader threads increases and can become a bottleneck in resource-constrained environments. Eventually, the number of Java threads created can cause out-of-memory exceptions if the number of concurrent requests exceeds the system's available resources.

To help address this potential problem, you can configure the ORB to use JNI reader threads where a finite number of reader threads, implemented using native OS threads instead of Java threads, are created during ORB initialization. JNI reader threads rely on the native OS TCP/IP asynchronous mechanism that enables a single native OS thread to handle I/O events from multiple sockets at the same time. The ORB manages the use of the JNI reader threads and assigns one of the available threads to handle the connection request, using a round-robin algorithm. Ordinarily, JNI reader threads should only be configured when using Java threads is too memory-intensive for your application environment.

The number of JNI reader threads you should allocate for an ORB depends on many factors and varies significantly from one environment to another, depending on available system resources and workload requirements. The following potential benefits might be achieved if you use JNI threads:

- Because a fixed number of threads is allocated, memory usage is reduced. This reduction provides significant benefit in environments with unusually large and sustained client-request workloads.
- The time needed to dynamically create and destroy Java threads is eliminated because a fixed number of JNI threads is created and allocated during ORB initialization.
- Each JNI thread can handle up to 1024 socket connections and interacts directly with the asynchronous I/O native OS mechanism, which might provide enhanced performance of network I/O processing.

Object Request Broker service settings

Use this page to configure the Java Object Request Broker (ORB) service.

To view this administrative console page, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Container services > ORB service**.

Several settings are available for controlling internal Object Request Broker (ORB) processing. You can use these settings to improve application performance in the case of applications that contain enterprise beans. You can make changes to these settings for the default server or any application server that is configured in the administrative domain.

Request timeout

Specifies the number of seconds to wait before timing out on a request message.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.RequestTimeout.

Data type	int
Units	Seconds
Default	180
Range	0 - largest integer recognized by Java

Request retries count

Specifies the number of times that the ORB attempts to send a request if a server fails. Retrying sometimes enables recovery from transient network failures. This field is ignored for z/OS.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.requestRetriesCount.

Data type	int
Default	1
Range	1 to 10

Request retries delay

Specifies the number of milliseconds between request retries. This field is ignored for z/OS.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.requestRetriesDelay.

Data type	int
Units	Milliseconds
Default	0
Range	0 to 60,000

Connection cache maximum

Specifies the maximum number of entries that can occupy the ORB connection cache before the ORB starts to remove inactive connections from the cache. This field is ignored for z/OS.

It is possible that the number of active connections in the cache will temporarily exceed this threshold value. If necessary, the ORB will continue to add connections as long as resources are available.

For use in command-line scripting, the full name of this system property is com.ibm.CORBA.MaxOpenConnections.

Data type	Integer
Units	Connections
Default	240
Range	10 - largest integer recognized by Java

Connection cache minimum

Specifies the minimum number of entries in the ORB connection cache. This field is ignored for z/OS.

The ORB will not remove inactive connections when the number of entries is below this value.

For use in command-line scripting, the full name of this system property is com.ibm.CORBA.MinOpenConnections.

Data type	Integer
Units	Connections
Default	100
Range	Any integer that is at least 5 less than the value specified for the Connection cache maximum property.

ORB tracing

Enables the tracing of ORB General Inter-ORB Protocol (GIOP) messages.

This setting affects two system properties: com.ibm.CORBA.Debug and com.ibm.CORBA.CommTrace. If you set these properties through command-line scripting, you must set both properties to `true` to enable the tracing of GIOP messages.

Data type	Boolean
Default	Not enabled (false)

Locate request timeout

Specifies the number of seconds to wait before timing out on a LocateRequest message. This field is ignored for z/OS.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.LocateRequestTimeout.

Data type	int
Units	Seconds
Default	180
Range	0 to 300

Force tunneling

Controls how the client ORB attempts to use HTTP tunneling. This field is ignored for z/OS.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.ForceTunnel.

Data type	String
Default	NEVER
Range	Valid values are ALWAYS, NEVER, or WHENREQUIRED.

Considering the following information when choosing the valid value:

ALWAYS
Use HTTP tunneling immediately, without trying TCP connections first.

NEVER
Disable HTTP tunneling. If a TCP connection fails, a CORBA system exception (COMM_FAILURE) occurs.

WHENREQUIRED
Use HTTP tunneling if TCP connections fail.

Tunnel agent URL

Specifies the Web address of the servlet to use in support of HTTP tunneling. This field is ignored on the z/OS platform.

This Web address must be a proper format:

http://w3.mycorp.com:81/servlet/com.ibm.CORBA.services.IIOPTunnelServlet

For applets: http://applethost:port/servlet/com.ibm.CORBA.services.IIOPTunnelServlet.

This field is required if HTTP tunneling is set. If you use command-line scripting, the full name of this system property is com.ibm.CORBA.TunnelAgentURL.

Pass by reference

Specifies how the ORB passes parameters. If enabled, the ORB passes parameters by reference instead of by value, to avoid making an object copy. If you do not enable the pass by reference option, a copy of the parameter passes rather than the parameter object itself. This can be expensive because the ORB must first make a copy of each parameter object.

You can use this option only when the Enterprise JavaBeans (EJB) client and the EJB are on the same classloader. This requirement means that the EJB client and the EJB must be deployed in the same EAR file.

If the Enterprise JavaBeans (EJB) client and server are installed in the same instance or the product, and the client and server use remote interfaces, enabling the pass by reference option can improve performance up to 50%. The pass by reference option helps performance only where non-primitive object types are passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Note: Enable this property with caution because unexpected behavior can occur. If an object reference is modified by the callee, the caller's object is modified as well, since they are the same object.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.iiop.noLocalCopies.

Data type	Boolean
Default	Not enabled (false)

The use of this option for enterprise beans with remote interfaces violates Enterprise JavaBeans (EJB) Specification, Version 2.0 (see section 5.4). Object references passed to Enterprise JavaBeans (EJB) methods or to EJB home methods are not copied and can be subject to corruption.

Consider the following example:

```
Iterator iterator = collection.iterator();
MyPrimaryKey pk = new MyPrimaryKey();
while (iterator.hasNext()) {
   pk.id = (String) iterator.next();
   MyEJB myEJB = myEJBHome.findByPrimaryKey(pk);
}
```

In this example, a reference to the same MyPrimaryKey object passes into the product with a different ID value each time. Running this code with pass by reference enabled causes a problem within the application server because multiple enterprise beans are referencing the same MyPrimaryKey object. To avoid this problem, set the com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation system property to `true` when the pass by reference option is enabled. Setting the pass by reference option to `true` causes the EJB container to make a local copy of the PrimaryKey object. As a result, however, a small portion of the performance advantage of setting the pass by reference option is lost.

As a general rule, any application code that passes an object reference as a parameter to an enterprise bean method or to an EJB home method must be scrutinized to determine if passing that object reference results in loss of data integrity or in other problems.

After examining your code, you can enable the pass by reference option by setting the com.ibm.CORBA.iiop.noLocalCopies system property to `true`. You can also enable the pass by reference option in the administrative console. Click **Servers > Server Types > Application servers >** *server_name* **> Container services > ORB Service** and select **Pass by reference**.

Object Request Broker custom properties

There are several ways to configure an Object Request Broker (ORB). For example, you can use ORB custom property settings, or system property settings to configure an ORB, or you can provide objects during ORB initialization. If you use the following ORB custom properties to configure an ORB, remember that two types of default values exist for some of these properties: the Java SE Development Kit (JDK) default values and the WebSphere Application Server default values.

The JDK default is the value that the ORB uses for a property if the property is not specified in any way. The WebSphere Application Server default is the value that the WebSphere Application Server product sets for a property in one of the following files:

- The orb.properties file when an application server is installed.
- The server.xml file when an application server is configured.

Because WebSphere Application Server explicitly sets its default value, if both a WebSphere Application Server and a JDK default value are defined for a property, the WebSphere Application Server default takes precedence over the JDK default.

For more information about the different ways to specify ORB properties and the precedence order, read the JDK Diagnostic Guide for the version of the JDK that you are using.

The `orb.properties` file, that is located in the *was_home*/java/jre/lib directory, contains ORB custom properties that are initially set to the WebSphere Application Server default values during the product installation process.

You can use the administrative console to specify new values for these ORB custom properties. Any value that you specify takes precedence over any JDK or WebSphere Application Server default values for these properties. The ORB custom properties settings that you specify in the administrative console are stored in the `server.xml` system file and are passed to an ORB in a properties object whenever an ORB is initialized.

To use the administrative console to set ORB custom properties, click **Servers > Server Types > Application servers >** *server_name* **> Container services > ORB service > Custom properties**. You can then change the setting of one of the listed custom properties or click **New** to add a new property to the list. Then, click **Apply** to save your change. When you finish making changes, click **OK** and then click **Save** to save your changes.

To use the java command on a command line, use the `-D` option; for example:

```
java -Dcom.ibm.CORBA.propname1=value1 -Dcom.ibm.CORBA.propname2=value2 ... application name
```

To use the launchclient command on a command line, prefix the property with `-CC`; for example:

```
launchclient yourapp.ear -CCDcom.ibm.CORBA.propname1=value1 -CCDcom.ibm.CORBA.propname2=value2
   ... optional application arguments
```

The Custom properties page might already include Secure Sockets Layer (SSL) properties that were added during product installation. A list of the additional properties that are associated with the ORB service follows. Unless otherwise indicated, the default values that are provided in the descriptions of these properties are the JDK default values.

com.ibm.CORBA.BootstrapHost

Specifies the domain name service (DNS) host name or IP address of the machine on which initial server contact for this client resides.

Note: This setting is deprecated.

For a command-line or programmatic alternative, read the topic *Client-side programming tips for the Object Request Broker service*.

com.ibm.CORBA.BootstrapPort

Specifies the port that the ORB uses to bootstrap to the machine on which the initial server contact for this client listens.

Note: This setting is deprecated.

For a command line or programmatic alternative, read the topic *Client-side programming tips for the Object Request Broker service*.

Default 2809

com.ibm.CORBA.ConnectTimeout

The com.ibm.CORBA.ConnectTimeout property specifies the maximum time, in second, that the client ORB waits before timing out when attempting to establish an IIOP connection with a remote server ORB. Typically, client applications use this property. By default, this property is not used by the application server. However, if necessary, you can specify the property for each individual application server through the administrative console.

Client applications can specify the com.ibm.CORBA.ConnectTimeout property in one of two ways:

- By including it in the orb.properties file
- By using the -CCD option to set the property with the launchclient script. The following example specifies a maximum timeout value of ten seconds:

  ```
  launchclient clientapp.ear -CCDcom.ibm.com.CORBA.ConnectTimeout=10...
  ```

Begin by setting your timeout value to 20-30 seconds, but consider factors such as network congestion and application server load and capacity. Lower values can provide better failover performance, but can result in exceptions if the remote server does not have enough time to complete the connection.

Valid Range	0-300
Default	0, which means that the client ORB waits indefinitely

com.ibm.CORBA.ConnectionInterceptorName

Specifies the connection interceptor class that is used to determine the type of outbound IIOP connection to use for a request, and if secure, the quality of protection characteristics associated with the request.

WebSphere Application Server default	com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
JDK default	None

com.ibm.CORBA.enableLocateRequest

Specifies whether the ORB uses the locate request mechanism to find objects in a WebSphere Application Server cell. Use this property for performance tuning.

When this property is set to `true`, the ORB first sends a short message to the server to find the object that it needs to access. This first contact is called the *locate request*. If most of your initial method invocations are small, setting this property to `false` might improve performance because this setting change can reduce the GIOP traffic by as much as one-half. If most of your initial method invocations are large, you should set this property to `true`. When the property is set to `true`, the small locate request message is sent instead of the large locate request message. The large message is then sent to the target after the desired object is found.

WebSphere Application Server default	true
JDK default	false

com.ibm.CORBA.FragmentSize

Specifies the size of GIOP fragments that the ORB uses when it sends requests. If the total size of a request exceeds the set value, the ORB breaks the request into fragments, and sends each fragment separately until the entire request is sent. Set this property on the client side with a -D system property if you use a stand-alone Java application.

Adjust the value specified for the com.ibm.CORBA.FragmentSize property if the amount of data that is sent over IIOP in most GIOP requests exceeds one kilobyte, or if thread dumps show that most client-side threads are waiting while sending or receiving data. Most messages should have few or no fragments.

If you want to instruct the ORB not to chunk any of the requests or replies it sends, set this property to `0`. However, setting the value to zero does not prevent the ORB from receiving GIOP fragments in requests or replies sent by another ORB.

Units	Bytes.
Default	1024
Range	From 64 to the largest value of a Java integer type that is divisible by 8

com.ibm.CORBA.ListenerPort

Specifies the port on which this server listens for incoming requests. This setting only applies for client-side ORBs.

Default	Next available system-assigned port number
Range	0 to 2147483647

com.ibm.CORBA.LocalHost

Specifies the host name or IP address of the system on which the server ORB is running. If you do not specify a value for this property, during ORB initialization, the product sets this property to the host name or IP address specified for the BOOTSTRAP_ADDRESS end point in the serverindex.xml file. For client applications, if no value is specified for this property, the ORB obtains a value at run time by calling InetAddress.getLocalHost().getHostAddress() method.

com.ibm.CORBA.numJNIReaders

Specifies the number of JNI reader threads to be allocated in the JNI reader thread pool that is used by the ORB. Each thread can handle up to 1024 connections.

Note: Before you specify this property, verify that a JSSE provider is selected as the provider for the SSL repertoire that is associated with the port on which the ORB service listens for incoming requests. You can specify either `IBMJSSE2 SSL` or `IBMJSSE SSL`. `IBMJSSE2 SSL` is the default provider setting for SSL repertoires.

Valid Range	1 - 2147483647
Default	4

com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl

Specifies that JNI reader threads are used. The property name specifies the class name of the ORB component that manages the pool of JNI reader threads and interacts with the native OS library used to process multiple connections simultaneously.

Note:

- Verify that the library is located in the bin directory for the product.

  For an Intel® operating system, the name of the file that contains the library name Selector.dll

  For a UNIX-based operating system, the name of the file that contains the library is either libSelector.a or libSelector.so. If the lib prefix is missing from the file name, rename the file such that the name includes the lib prefix.

  For the UNIX platform,
- When you specify this property using the administrative console, enter `com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl` for the property name and an empty string (`""`) for the value.

  When you specify this property on the java command, do not include a value:

  ```
  -Dcom.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.JNIReaderPoolImpl
  ```

Valid Range	Not applicable
Default	None

com.ibm.CORBA.RasManager

Specifies an alternative to the default RAS manager of the ORB. This property must be set to com.ibm.websphere.ras.WsOrbRasManager before the ORB can be integrated with the rest of the RAS processing for the product.

WebSphere Application Server default	com.ibm.websphere.ras.WsOrbRasManager
JDK default	None

com.ibm.CORBA.ServerSocketQueueDepth

Specifies the maximum number of connection requests that can be waiting to be handled by the Server ORB before the product starts to reject new incoming connection requests. This property corresponds to the `backlog` argument to a ServerSocket constructor and is handled directly by TCP/IP.

If you see a ″connection refused″ message in a trace log, typically, either the port on the target machine is not open, or the server is overloaded with queued-up connection requests. Increasing the value specified for this property can help alleviate this problem if there does not appear to be any other problem in the system.

Default	50
Range	From 50 to the largest value of the Java int type

com.ibm.CORBA.ShortExceptionDetails

Specifies that the exception detail message that is returned whenever the server ORB encounters a CORBA system exception contains a short description of the exception as returned by the toString method of java.lang.Throwable class. Otherwise, the message contains the complete stack trace as returned by the printStackTrace method of java.lang.Throwable class.

com.ibm.CORBA.WSSSLClientSocketFactoryName

Specifies the class that the ORB uses to create SSL sockets for secure outbound IIOP connections.

WebSphere Application Server default	com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl
JDK default	None

com.ibm.CORBA.WSSSLServerSocketFactoryName

Specifies the class that the ORB uses to create SSL sockets for inbound IIOP connections.

WebSphere Application Server default	com.ibm.ws.security.orbssl.WSSSLServerSocketFactoryImpl
JDK default	None

com.ibm.websphere.threadpool.strategy.implementation

Specifies the logical pool distribution (LPD) thread pool strategy that takes affect the next time you start the application server, and is enabled if set to com.ibm.ws.threadpool.strategy.LogicalPoolDistribution.

Note: The logical pool distribution function is deprecated. Do not configure logical pool distribution unless you have already configured it for a previous release of the product.

Some requests have shorter start times than others. LPD is a mechanism for providing these shorter requests more access to start threads. For more information, read the topic *Logical pool distribution*, that is included in the product information center.

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.calcinterval

Specifies how often the logical pool distribution (LPD) mechanism readjusts the pool start target times. This property cannot be turned off after this support is installed.

Note: The logical pool distribution function is deprecated. Do not configure logical pool distribution unless you have already configured it with a previous release of the product.

If you use this property, LPD must be enabled. Read the description of the com.ibm.websphere.threadpool.strategy.implementation property for more information.

Data type	Integer
Units	Milliseconds
Default	30
Range	20,000 milliseconds minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.lruinterval

Specifies, in milliseconds, how long the logical pool distribution internal data is kept for inactive requests. This mechanism tracks several statistics for each request type that is received. Consider removing requests that have been inactive for an unusually long length of time.

Note: This function is deprecated. Do not configure logical pool distribution unless you have already configured it with a previous release of the product.

If you use this property, LPD must be enabled. Read the description of the com.ibm.websphere.threadpool.strategy.implementation property for more information.

Data type	Integer
Units	Milliseconds
Default	300000 (5 minutes)
Range	60000 (1 minute) minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.outqueues

Specifies how many pools are created and how many threads are allocated to each pool in the logical pool distribution mechanism.

Note: The logical pool distribution function is deprecated. Do not configure logical pool distribution unless you have already configured it with a previous release of the product.

The ORB parameter for specifying the maximum number of threads controls the total number of threads. The outqueues parameter is specified as a comma separated list of percentages that add up to 100. For example, the list 25,25,25,25 sets up 4 pools, each allocated 25 percent of the available ORB thread pool. The pools are indexed left to right from 0 to n-1. The calculation mechanism dynamically assigns each outqueue a target start time. Target start times are assigned to outqueues in increasing order. Therefore, pool 0 gets the requests with the least start time, and pool n-1 gets requests with the highest start times.

If you specify this property, LPD must be enabled. Read the description of the com.ibm.websphere.threadpool.strategy.implementation property for more information.

Data type	Integers in comma separated list
Default	25,25,25,25
Range	Percentages in list must total 100 percent

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.statsinterval

Specifies that statistics are dumped to stdout after this interval expires, but only if requests are processed. This process keeps the mechanism from filling the log files with redundant information. These statistics are beneficial for tuning the logical pool distribution mechanism.

Note: The logical pool distribution function is deprecated. Do not configure logical pool distribution unless you have already configured it with a previous release of the product.

If you use this property, LPD must be enabled. Read the description of the com.ibm.websphere.threadpool.strategy.implementation property for more information.

Data type	Integer
Units	Milliseconds
Default	0 (off)
Range	30,000 (30 seconds) minimum

com.ibm.websphere.threadpool.strategy.LogicalPoolDistribution.workqueue

Specifies the size of a new queue where incoming requests wait for dispatch. Pertains to the logical pool distribution mechanism.

Note: The logical pool distribution function is deprecated. Do not configure logical pool distribution unless you have already configured it with a previous release of the product.

If you use this property, LPD must be enabled. Read the description of the com.ibm.websphere.threadpool.strategy.implementation property for more information.

Data type	Integer
Default	96
Range	10 minimum

com.ibm.ws.orb.services.redirector.MaxOpenSocketsPerEndpoint

Specifies the maximum number of connections that the IIOP Tunnel Servlet maintains in its connection cache for each target host and port. If the number of concurrent client requests to a single host and port exceeds the setting for this property, the IIOP Tunnel Servlet opens a temporary connection to the target server for each extra client request, and then closes the connection after it receives the reply. Connections that are opened, but not used within five minutes, are removed from the cache for the IIOP Tunnel Servlet.

WebSphere Application Server default	3
JDK default	Not applicable
Range	0 - largest integer recognized by Java

com.ibm.ws.orb.services.redirector.RequestTimeout

Specifies the number of seconds that the IIOP Tunnel Servlet waits for a reply from the target server on behalf of a client before timing out. If a value is not specified for this property, or is incorrectly specified, the com.ibm.CORBA.RequestTimeout property setting for the application server, on which the IIOP Tunnel Servlet is installed, is used as the setting for the com.ibm.ws.orb.services.redirector.RequestTimeout property.

The value you specify for this property must be at least as high as the highest client setting for the com.ibm.CORBA.RequestTimeout property; otherwise the IIOP Tunnel Servlet might timeout more quickly than the client typically times out while waiting for a reply. If this property is set to zero, the IIOP Tunnel Servlet does not time out.

WebSphere Application Server default	com.ibm.CORBA.RequestTimeout property setting for the application server on which the IIOP Tunnel Servlet is installed.
JDK default	Not applicable
Range	0 - largest integer recognized by Java

com.ibm.ws.orb.transport.useMultiHome

Specifies whether the server ORB binds to all network interfaces in the system. If you specify `true`, the ORB binds to all network interfaces that are available to it. If you specify `false`, the ORB only binds to the network interface that is specified for the com.ibm.CORBA.LocalHost system property.

WebSphere Application Server default	true
JDK default	true

javax.rmi.CORBA.UtilClass

Specifies the name of the Java class that the product uses to implement the javax.rmi.CORBA.UtilDelegate interface.

This property supports delegation for method implementations in the javax.rmi.CORBA.Util class. The javax.rmi.CORBA.Util class provides utility methods that can be used by stubs and ties to perform common operations. The delegate is a singleton instance of a class that implements this interface and provides a replacement implementation for all of the methods of javax.rmi.CORBA.Util. To enable a delegate, provide the class name of the delegate as the value of the javax.rmi.CORBA.UtilClass system property. The default value provides support for the com.ibm.CORBA.iiop.noLocalCopies property.

WebSphere Application Server default	com.ibm.ws.orb.WSUtilDelegateImpl
JDK default	None

Object Request Broker communications trace

The Object Request Broker (ORB) communications trace, typically referred to as *CommTrace*, contains the sequence of General InterORB Protocol (GIOP) messages sent and received by the ORB when the application is running.

It might be necessary to understand the low-level sequence of client-to-server or server-to-server interactions during problem determination. This topic uses trace entries from log examples to explain the contents of the log and help you understand the interaction sequence. It focuses only in the GIOP messages and does not discuss in detail additional trace information that displays when intervening with the GIOP-message boundaries.

Location

When ORB tracing is enabled, this information is placed in the *profile_root*/logs/*server_name*/trace.log directory.

About the ORB trace file

The following are properties of the file that is created when ORB tracing is enabled.
- Read-only
- Updated by the administrative function
- Use this file to localize and resolve ORB-related problems.

How to interpret the output

The following sections refer to sample log output found later in this topic.

Identifying information

The start of a GIOP message is identified by a line that contains either OUT GOING: or IN COMING: depending on whether the message is sent or received by the process.

Following the identifying line entry is a series of items, formatted for convenience, with information extracted from the raw message that identifies the endpoints in this particular message interaction. See lines 3-13 in both examples. The formatted items include the following:
- GIOP message type (line 3)
- Date and time that the message was recorded (line 4)
- Information that is useful to identify the thread that is running when the message records, and other thread-specific information (line 5)
- Local and remote TCP/IP ports used for the interaction (lines 6 through 9)
- GIOP version, byte order, an indication of whether the message is a fragment, and message size (lines 10 through 13)

Request ID, response expected and reply status

Following the introductory message information, the request ID is an integer generated by the ORB. It is used to identify and associate each request with its corresponding reply. This association is necessary because the ORB can receive requests from multiple clients and must be able to associate each reply with the corresponding originating request.

- Lines 15-17 in the request example show the request ID, followed by an indication to the receiving endpoint that a response is expected (CORBA supports sending one-way requests for which a response is not expected.)
- Line 15 in the *Sample Log Entry - GIOP Reply* shows the request ID; line 33 shows the reply status received after completing the previously sent request.

Object Key

Lines 18-20 in the request example show the object key, the internal representation used by the ORB to identify and locate the target object intended to receive the request message. Object keys are not standardized.

Operation

Line 21 in the request example shows the name of the operation that the target object starts in the receiving endpoint. In this example, the specific operation requested is named _get_value.

Service context information

The service contexts in the message are also formatted for convenience. Each GIOP message might contain a sequence of service contexts sent and received by each endpoint. Service contexts, identified uniquely with an ID, contain data used in the specific interaction, such as security, character code set conversion, and ORB version information. The content of some of the service contexts is standardized and specified by OMG, while other service contexts are proprietary and specified by each vendor. IBM-specific service contexts are identified with IDs that begin with 0x4942.

Lines 22-41 in the request example illustrate typical service context entries. Three service contexts are in the request message, as shown in line 22. The ID, length of data, and raw data for each service context is printed next. Lines 23-25 show an IBM-proprietary context, as indicated by the 0x49424D12 ID. Lines 26-41 show two standard service contexts, identified by 0x6 ID (line 26) and the 0x1 ID (line 39).

Lines 16-32 in the *Sample Log Entry - GIOP Reply* illustrate two service contexts, one IBM-proprietary (line 17) and one standardized (line 20).

For the definition of the standardized service contexts, see the CORBA specification. Service context 0x1 (CORBA::IOP::CodeSets) is used to publish the character code sets supported by the ORB in order to negotiate and determine the code set used to transmit character data. Service context 0x6 (CORBA::IOP::SendingContextRunTime) is used by Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP) to provide the receiving endpoint with the IOR for the SendingContextRuntime object. IBM service context 0x49424D12 is used to publish ORB PartnerVersion information to support release-to-release interoperability between sending and receiving ORBs.

Data offset

Line 42 in the request example shows the offset, relative to the beginning of the GIOP message, where the remainder body of the request or reply message is located. This portion of the message is specific to each operation and varies from operation to operation. Therefore, it is not formatted, as the specific contents are not known by the ORB. The offset is printed as an aid to quickly locating the operation-specific data in the raw GIOP message dump, which follows the data offset.

Raw GIOP message dump

Starting at line 45 in the request example and line 36 in the *Sample Log Entry - GIOP Reply*, a raw dump of the entire GIOP message is printed in hexadecimal format. Request messages contain the parameters required by the given operation and reply messages contain the return values and content of output parameters as required by the given operation. For brevity, not all of the raw data is in the figures.

Sample Log Entry - GIOP Request

```
1.  OUT GOING:

3.  Request Message
4.  Date:          April 17, 2002 10:00:43 PM CDT
5.  Thread Info:   P=842115:O=1:CT
6.  Local Port:    1243 (0x4DB)
```

```
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10.  GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  268 (0x10C)
--
15. Request ID:        5
16. Response Flag:     WITH_TARGET
17. Target Address:     0
18. Object Key:        length = 24 (0x18)
                       4B4D4249 00000010 BA4D6D34 000E0008
                       00000000 00000000
21. Operation:         _get_value
22. Service Context:   length = 3 (0x3)
23. Context ID:  1229081874 (0x49424D12)
24. Context data:  length = 8 (0x8)
                       00000000 13100003
26. Context ID:  6 (0x6)
27. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
                       6F72672F 53656E64 696E6743 6F6E7465
                       78742F43 6F646542 6173653A 312E3000
                       00000001 00000000 00000068 00010200
                       0000000E 3139322E 3136382E 312E3130
                       310004DC 00000018 4B4D4249 00000010
                       BA4D6D69 000E0008 00000000 00000000
                       00000002 00000001 00000018 00000000
                       00010001 00000001 00010020 00010100
                       00000000 49424D0A 00000008 00000000
                       13100003
39. Context ID:  1 (0x1)
40. Context data:  length = 12 (0xC)
                       00000000 00010001 00010100
42. Data Offset:       118


45. 0000: 47494F50 01020000 0000010C 00000005   GIOP............
46. 0010: 03000000 00000000 00000018 4B4D4249   ............KMBI
47. 0020: [remainder of message body deleted for brevity]
```

Sample Log Entry - GIOP Reply

```
1.  IN COMING:

3.  Reply Message
4.  Date:          April 17, 2002 10:00:47 PM CDT
5.  Thread Info:   RT=0:P=842115:O=1:com.ibm.rmi.transport.TCPTransportConnection
5a (line 5 broken for publication).   remoteHost=192.168.1.101 remotePort=1242 localPort=1243
6.  Local Port:    1243 (0x4DB)
7.  Local IP:      jdoe.austin.ibm.com/192.168.1.101
8.  Remote Port:   1242 (0x4DA)
9.  Remote IP:     jdoe.austin.ibm.com/192.168.1.101
10. GIOP Version:  1.2
11. Byte order:    big endian
12. Fragment to follow: No
13. Message size:  208 (0xD0)
--
15. Request ID:        5
16. Service Context:   length = 2 (0x2)
17. Context ID:  1229081874 (0x49424D12)
18. Context data:  length = 8 (0x8)
                       00000000 13100003
20. Context ID:  6 (0x6)
21. Context data:  length = 164 (0xA4)
                       00000000 00000028 49444C3A 6F6D672E
```

```
                        6F72672F 53656E64 696E6743 6F6E7465
                        78742F43 6F646542 6173653A 312E3000
                        00000001 00000000 00000068 00010200
                        0000000E 3139322E 3136382E 312E3130
                        310004DA 00000018 4B4D4249 00000010
                        BA4D6D34 000E0008 00000001 00000000
                        00000002 00000001 00000018 00000000
                        00010001 00000001 00010020 00010100
                        00000000 49424D0A 00000008 00000000
                        13100003
33. Reply Status:       NO_EXCEPTION


36. 0000: 47494F50 01020001 000000D0 00000005   GIOP............
37. 0010: 00000000 00000002 49424D12 00000008   ........IBM.....
38. 0020: [remainder of message body deleted for brevity]
```

Client-side programming tips for the Object Request Broker service

Every Internet InterORB Protocol (IIOP) request and response exchange consists of a client-side ORB and a server-side ORB. It is important that any application that uses IIOP is properly programmed to communicate with the client-side Object Request Broker (ORB).

The following tips should help you ensure that an application that uses IIOP to handle request and response exchanges is properly programmed to communicate with the client-side Object Request Broker (ORB).

Resolution of initial references to services

Client applications can use the ORBInitRef and ORBDefaultInitRef properties to configure the network location that the ORB service uses to find a service such as naming. When set, these properties are included in the parameters that are used to initialize the ORB, as illustrated in the following example:

```
org.omg.CORBA.ORB.init(java.lang.String[] args,
                       java.util.Properties props)
```

You can set these properties in client code or by command-line argument. It is possible to specify more than one service location by using multiple ORBInitRef property settings (one for each service), but only a single ORBDefaultInitRef value can be specified.

For setting in client code, these properties are com.ibm.CORBA.ORBInitRef.*service_name* and com.ibm.CORBA.ORBDefaultInitRef, respectively. For example, to specify that the naming service (NameService) is located in sample.server.com at port 2809, set the com.ibm.CORBA.ORBInitRef.NameService property to corbaloc::sample.server.com:2809/NameService.

For setting by command-line argument, these properties are -ORBInitRef and -ORBDefaultInitRef, respectively. To locate the same naming service specified previously, use the following Java command:

After these properties are set for services that the ORB supports, Java Platform, Enterprise Edition (Java EE) applications can call the resolve_initial_references function on the ORB, as defined in the CORBA/IIOP specification, to obtain the initial reference to a given service.

Preferred API for obtaining an ORB instance

For Java EE applications, you can use either of the following approaches. However, it is strongly recommended that you use the Java Naming and Directory Interface (JNDI) approach to ensure that the same ORB instance is used throughout the client application; you avoid the unintended inconsistencies that might occur when different ORB instances are used.

JNDI approach: For Java EE applications (including enterprise beans, Java EE clients and servlets), you can obtain an ORB instance by creating a JNDI InitialContext object and looking up the ORB under the java:comp/ORB name, as illustrated in the following example:

```
javax.naming.Context ctx = new javax.naming.InitialContext();
org.omg.CORBA.ORB orb =
   (org.omg.CORBA.ORB)javax.rmi.PortableRemoteObject.narrow(ctx.lookup("java:comp/ORB"),
                                                            org.omg.CORBA.ORB.class);
```

The ORB instance obtained using JNDI is a singleton object, shared by all the Java EE components that are running in the same Java virtual machine process.

CORBA approach: Because thin-client applications do not run in a Java EE container, they cannot use JNDI interfaces to look up the ORB. In this case, you can obtain an ORB instance by using CORBA programming interfaces, as follows:

```
java.util.Properties props = new java.util.Properties();
java.lang.String[] args = new java.lang.String[0];
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, props);
```

In contrast to the JNDI approach, the CORBA specification requires that a new ORB instance be created each time the ORB.init method is called. If necessary to change the ORB default settings, you can add ORB property settings to the Properties object that is passed in the ORB.init method call.

The use of the com.ibm.ejs.oa.EJSORB.getORBinstance method, supported in previous releases of this product is deprecated.

API restrictions with sharing an ORB instance among Java EE application components

For performance reasons, it often makes sense to share a single ORB instance among components in a Java EE application. As required by the Java EE Specification, Version 1.3, all Web and EJB containers provide an ORB instance in the JNDI namespace as java:comp/ORB. Each container can share this instance among application components but is not required to. For proper isolation between application components, application code must comply with the following restrictions:

- Do not call the ORB shutdown or destroy methods
- Do not call org.omg.CORBA_2_3.ORB methods register_value_factory, or unregister_value_factory

In addition, do not share an ORB instance among application components in different Java EE applications.

Required use of rmic and idlj that ship with the IBM Developer Kit

The Java Runtime Environment (JRE) used by this product includes the **rmic** and **idlj** tools. You use the tools to generate Java language bindings for the CORBA/IIOP protocol.

During product installation, the tools are installed in the *app_server_root*/java/ibm_bin directory. Versions of these tools included with Java development kits in the $JAVA_HOME/bin directory other than the IBM Developer Kit installed with this product are incompatible with this product.

When you install this product, the *app_server_root*/java/ibm_bin directory is included in the $PATH search order to enable use of the rmic and idlj scripts provided by IBM. Because the scripts are in the *app_server_root*/java/ibm_bin directory instead of the JRE standard *app_server_root*/java/bin directory, it is unlikely that you can overwrite them when applying maintenance to a JRE not provided by IBM.

In addition to the rmic and idlj tools, the JRE also includes Interface Definition Language (IDL) files. The files are based on those defined by the Object Management Group (OMG) and can be used by applications that need an IDL definition of selected ORB interfaces. The files are placed in the *app_server_root*/java/ibm_lib directory.

Before using either the rmic or idlj tool, ensure that the *app_server_root*/java/ibm_bin directory is included in the proper PATH variable search order in the environment. If your application uses IDL files in the *app_server_root*/java/ibm_lib directory, also ensure that the directory is included in the PATH variable.

Character code set conversion support for the Java Object Request Broker service

The CORBA/IIOP specification defines a framework for negotiation and conversion of character code sets used by the Java Object Request Broker (ORB) service.

This product supports the framework and provides the following system properties for modifying the default settings:

com.ibm.CORBA.ORBCharEncoding

Specifies the name of the native code set that the ORB uses for character data (referred to as *NCS-C* in the CORBA/IIOP specification). By default, the ORB uses UTF8. Valid code set values for this property are shown in the table that follows this list; values that are valid only for ORBWCharDefault are indicated.

com.ibm.CORBA.ORBWCharDefault

Specifies the default code set that the ORB uses for transmission of wide character data when no code set for wide character data is found in the tagged component in the Interoperable Object Reference (IOR) or in the GIOP service context. If no code set for wide character data is found and this property is not set, the ORB raises an exception, as specified in the CORBA specification. No default value is set for this property. The only valid code set values for this property are UCS2 or UTF16.

The CORBA code set negotiation and conversion framework specifies the use of code set registry IDs as defined in the Open Software Foundation (OSF) code set registry. The ORB translates the Java file.encoding names shown in the following table to the corresponding OSF registry IDs. These IDs are then used by the ORB in the IOR Code set tagged component and GIOP code set service context as specified in the CORBA and IIOP specification.

Java name	OSF registry ID	Comments
ASCII	0x00010020	
ISO8859_1	0x00010001	
ISO8859_2	0x00010002	
ISO8859_3	0x00010003	
ISO8859_4	0x00010004	
ISO8859_5	0x00010005	
ISO8859_6	0x00010006	
ISO8859_7	0x00010007	
ISO8859_8	0x00010008	
ISO8859_9	0x00010009	
ISO8859_15_FDIS	0x0001000F	
Cp1250	0x100204E2	
Cp1251	0x100204E3	
Cp1252	0x100204E4	
Cp1253	0x100204E5	
Cp1254	0x100204E6	
Cp1255	0x100204E7	
Cp1256	0x100204E8	
Cp1257	0x100204E9	
Cp943C	0x100203AF	
Cp943	0x100203AF	

Java name	OSF registry ID	Comments
Cp949C	0x100203B5	
Cp949	0x100203B5	
Cp1363C	0x10020553	
Cp1363	0x10020553	
Cp950	0x100203B6	
Cp1381	0x10020565	
Cp1386	0x1002056A	
EUC_JP	0x00030010	
EUC_KR	0x0004000A	
EUC_TW	0x00050010	
Cp964	0x100203C4	
Cp970	0x100203CA	
Cp1383	0x10020567	
Cp33722C	0x100283BA	
Cp33722	0x100283BA	
Cp930	0x100203A2	
Cp1047	0x10020417	
UCS2	0x00010100	Valid only for the ORBWCharDefault
UTF8	0x05010001	
UTF16	0x00010109	Valid only for the ORBWCharDefault

Object Request Brokers: Resources for learning

Use the following links to find relevant supplemental information about Object Request Brokers (ORBs). The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- "Planning, business scenarios, and IT architecture"
- "Administration" on page 2106
- "Programming specifications" on page 2106

Planning, business scenarios, and IT architecture

- CORBA FAQ

  Getting started with Object Request Brokers and CORBA.
- WebSphere Application Server CORBA Interoperability

  This document describes WebSphere CORBA interoperability for WebSphere Application Server products.
- CORBA Interoperability Samples

  These samples demonstrate the general principles by which WebSphere Application Server applications can interoperate with CORBA applications.

Administration

- IANA Character Set Registry

  This document contains a list of all valid character encoding schemes.
- developerWorks WebSphere

Programming specifications

- Catalog Of OMG CORBA/IIOP Specifications

  This document provides a catalog of OMG CORBA/IIOP specifications.

Object request broker troubleshooting tips

Use these tips to diagnose problems related to the WebSphere Application Server Object Request Broker (ORB).

Enabling tracing for the Object Request Broker component

The object request broker (ORB) service is one of the product run time services. Tracing messages sent and received by the ORB is a useful starting point for troubleshooting the ORB service. You can selectively enable or disable tracing of ORB messages for each server in a product installation, and for each application client.

This tracing is referred to by the product support as a *comm trace*, and is different from the general purpose trace facility. The trace facility, which shows the detailed run-time behavior of product components, may be used alongside comm trace for other product components, or for the ORB component. The trace string associated with the ORB service is `ORBRas=all=enabled`.

You can enable and disable comm tracing using the administrative console or by manually editing the server.xml file for the server be trace. You must stop and restart the server for the configuration change to take effect.

For example, using the administrative console:

- Click **Servers > Server Types > WebSphere application servers >** *server_name* **> Container services > ORB service**, and then select the ORB tracing. Click **OK**, and then click **Save** to save your settings. Restart the server for the new settings to take effect. Or,
- Locate the server.xml file for the selected server, for example: *profile_root*/config/cells/*node_name*/nodes/*node_name*/servers/*server_name*/server.xml.
- Locate the services entry for the ORB service, `xmi:type=orb:ObjectRequestBroker`, and set `commTraceEnabled=true`.

ORB tracing for client applications requires that both the ORBRas component trace and the ORB comm trace are enabled. If only the ORB comm trace is enabled, no trace output is generated for client-side ORB traces. You can use one of the following options to enable both traces in the command line used to launch the client application:

- If you are using the product launcher, launchClient, use the **-CCD** option.
- If you are using the **java** command specify these parameters:

```
-trace=ORBRas=all=enabled
-tracefile=filename
-Dcom.ibm.CORBA.Debug=true
-Dcom.ibm.CORBA.CommTrace=true
```

ORB tracing output for thin clients can be directed by setting the `com.ibm.CORBA.Debug.Output = debugOutputFilename` parameter in the command line.

Note: When using launchClient on operating systems like AIX or Linux, because ORB tracing is integrated with the WebSphere Application Server general component trace, both the component and comm ORB traces are written to the same file as the rest of the WebSphere Application Server traces. The name of this file is specified on the `-CCtracefile=`*filename* parameter.

When using launchClient on a Windows operating systems, you get a separate ORB trace file, called orbtrc.*timestamp*.txt. This file is located in the current directory.

Log files and messages associated with Object Request Broker

Messages and trace information for the ORB are saved in the following logs:
- The *profile_root*/logs/*server_name*/trace.log file for output from communications tracing and tracing the behavior of the ORBRas component
- The JVM logs for each application server, for WebSphere Application Server error and warning messages

The following message in the SystemOut.log file indicates the successful start of the application server and its ORB service:

WSVR0001I: Server server1 open for e-business

When communications tracing is enabled, a message similar to the following example in the *profile_root*/logs/*server_name*/trace.log file, indicates that the ORB service has started successfully. The message also shows the start of a listener thread, which is waiting for requests on the specified local port.

com.ibm.ws.orbimpl.transport.WSTransport startListening(ServerConnectionData connectionData) P=693799:O=0:CT a new ListenerThread has been started for ServerSocket[addr=0.0.0.0/ 0.0.0.0,port=0,localport=1360]

When the `com.ibm.ejs.oa.*=all=enabled` parameter is specified, tracing of the Object Adapter is enabled. The following message in the trace.log indicates that the ORB service started successfully:

EJSORBImpl < initializeORB

The ORB service is one of the first services started during the WebSphere Application Server initialization process. If it is not properly configured, other components such as naming, and security are not likely to start successfully. This is obvious in the JVM logs or trace.log of the affected application server.

The following message in the SystemOut.log file is a warning message that indicates, in a future release, the ORB might use the thread pool that is defined in the Thread Pool Manager service only. If this situation occurs, you will not be able to define a thread pool in the Object Request Broker service.

WSVR0626W: The ThreadPool setting on the ObjectRequestBroker service is deprecated.

An action is not required when you receive this warning message. To prevent the warning message from reoccurring, configure the ORB to use the thread pool that is defined in the Thread Pool Manager service. Complete the following steps in the administrative console:

1. Click **Servers > Server Types > WebSphere application servers >** *server_name*.
2. In the Container Settings section, expand Container Services and click **ORB Service**
3. In the Thread Pool Settings section, select the option **Use the ORB.thread.pool settings associated with the Thread Pool Manager (recommended).** .
4. Click **OK** and **Save** to save your changes directly to the master configuration.

Adjusting object request broker timeout values

If Web clients that access Java applications running in the product environment are consistently experiencing problems with their requests, and the problem cannot be traced to other sources and addressed through other solutions, consider setting an ORB timeout value and adjusting it for your environment. A list of timeout scenarios follows:

- Web browsers vary in their language for indicating that they have timed out. Usually, the problem is announced as a connection failure or a no-path-to-server message.
- Set an ORB timeout value to less than the time after which a Web client eventually times out. Because it can be difficult to tell how long Web clients wait before timing out, setting an ORB timeout value requires experimentation. The ideal testing environment features some simulated network failures for testing the proposed setting value.
- Empirical results from limited testing indicate that 30 seconds is a reasonable starting value. Ensure that this setting is not too low. To fine tune the setting, find a value that is not too low. Gradually decrease the setting until reaching the threshold at which the value becomes too low. Set the value a little higher than the threshold.
- When an ORB timeout value is set too low, the symptom is numerous CORBA NO_RESPONSE exceptions, which occur even for some valid requests. The value is likely to be too low if requests that should have been successful, for example, the server is not down, are being lost or refused.

Timeout adjustments: Do not adjust an ORB timeout value unless you have a problem. Configuring a value that is inappropriate for the environment can create a problem. An incorrect value can produce results worse than the original problem.

You can adjust timeout intervals for the product Java ORB through the following administrative settings:

- **Request timeout**, the number of seconds to wait before timing out on most pending ORB requests if the network fails
- **Locate request timeout**, the number of seconds to wait before timing out on a locate-request message

Java packages containing the Object Request Broker service

The ORB service resides in the following Java packages:

- com.ibm.com.CORBA.*
- com.ibm.rmi.*
- com.ibm.ws.orb.*
- com.ibm.ws.orbimpl.*
- org.omg.CORBA.*
- javax.rmi.CORBA.*

JAR files that contain the previously mentioned packages include:

- *app_server_root*/java/jre/lib/ext/ibmorb.jar
- *app_server_root*/java/jre/lib/ext/iwsorbutil.jar
- *app_server_root*/lib/iwsorb.jar

Tools used with Object Request Broker

The tools used to compile Java remote interfaces to generate language bindings used by the ORB at runtime reside in the following Java packages:

- com.ibm.tools.rmic.*
- com.ibm.idl.*

The JAR file that contains the packages is *app_server_root*/java/lib/ibmtools.jar.

Object Request Broker properties

The ORB service requires a number of ORB properties for correct operation. It is not necessary for most users to modify these properties, and it is recommended that only your system administrator modify them when required.. Consult IBM Support personnel for assistance. The properties reside in the orb.properties file, located in *app_server_root*/java/jre/lib/orb.properties.

CORBA minor codes

The CORBA specification defines standard minor exception codes for use by the ORB when a system exception is thrown. In addition, the object management group (OMG) assigns each vendor a unique prefix value, called the Vendor Minor code ID (VMCID). This prefix value is used in vendor-proprietary minor exception codes. The VMCID that is assigned to OMG is 0x4f4d0, and the VMCID that is assigned to IBM is 0x4942F

Minor code values that are assigned to IBM, and that are used by the ORB in the product follow. The minor code value is in decimal and hexadecimal formats. The column labeled minor code reason gives a short description of the condition causing the exception.

Currently there is no documentation for these errors beyond the minor code reason. If you require technical support from IBM, the minor code helps support engineers determine the source of the problem.

The minor exception codes that are listed in the following table are in the format <VMCID><minor_code>. For example, the 4942f in the minor exception code 0x4942f102 indicates that this exception is an IBM-defined exception. The 102 at the end of this minor exception code is the actual minor code.

Table 38. Decimal minor exception codes 1229066320 to 1229066364

Decimal	Hexadecimal	Minor code reason
1229066320	0x49421050	HTTP_READER_FAILURE
1229066321	0x49421051	COULD_NOT_INSTANTIATE_CLIENT_SSL_SOCKET_FACTORY
1229066322	0x49421052	COULD_NOT_INSTANTIATE_SERVER_SSL_SOCKET_FACTORY
1229066323	0x49421053	CREATE_LISTENER_FAILED_1
1229066324	0x49421054	CREATE_LISTENER_FAILED_2
1229066325	0x49421055	CREATE_LISTENER_FAILED_3
1229066326	0x49421056	CREATE_LISTENER_FAILED_4
1229066327	0x49421057	CREATE_LISTENER_FAILED_5
1229066328	0x49421058	INVALID_CONNECTION_TYPE
1229066329	0x49421059	HTTPINPUTSTREAM_NO_ACTIVEINPUTSTREAM
1229066330	0x4942105a	HTTPOUTPUTSTREAM_NO_OUTPUTSTREAM
1229066331	0x4942105b	CONNECTIONINTERCEPTOR_INVALID_CLASSNAME
1229066332	0x4942105c	NO_CONNECTIONDATA_IN_CONNECTIONDATACARRIER
1229066333	0x4942105d	CLIENT_CONNECTIONDATA_IS_INVALID_TYPE
1229066334	0x4942105e	SERVER_CONNECTIONDATA_IS_INVALID_TYPE
1229066335	0x4942105f	NO_OVERLAP_OF_ENABLED_AND_DESIRED_CIPHER_SUITES
1229066352	0x49421070	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_CLIENT_SOCKET
1229066353	0x49421071	GETCONNECTION_KEY_RETURNED_FALSE
1229066354	0x49421072	UNABLE_TO_CREATE_SSL_SOCKET
1229066355	0x49421073	SSLSERVERSOCKET_TARGET_SUPPORTS_LESS_THAN_1
1229066356	0x49421074	SSLSERVERSOCKET_TARGET_REQUIRES_LESS_THAN_1
1229066357	0x49421075	SSLSERVERSOCKET_TARGET_LESS_THAN_TARGET_REQUIRES
1229066358	0x49421076	UNABLE_TO_CREATE_SSL_SERVER_SOCKET
1229066359	0x49421077	CAUGHT_EXCEPTION_WHILE_CONFIGURING_SSL_SERVER_SOCKET
1229066360	0x49421078	INVALID_SERVER_CONNECTION_DATA_TYPE
1229066361	0x49421079	GETSERVERCONNECTIONDATA_RETURNED_NULL

Table 38. Decimal minor exception codes 1229066320 to 1229066364 (continued)

1229066362	0x4942107a	GET_SSL_SESSION_RETURNED_NULL
1229066363	0x4942107b	GLOBAL_ORB_EXISTS
1229066364	0x4942107c	CONNECT_TIME_OUT

Table 39. Decimal minor exception codes 1229123841 to 1229124249

Decimal	Hexadecimal	Minor code reason
1229123841	0x4942f101	DSIMETHOD_NOTCALLED
1229123842	0x4942f102	BAD_INV_PARAMS
1229123843	0x4942f103	BAD_INV_RESULT
1229123844	0x4942f104	BAD_INV_CTX
1229123845	0x4942f105	ORB_NOTREADY
1229123879	0x4942f127	PI_NOT_POST_INIT
1229123880	0x4942f128	INVALID_EXTENDED_PI_CALL
1229123969	0x4942f181	BAD_OPERATION_EXTRACT_SHORT
1229123970	0x4942f182	BAD_OPERATION_EXTRACT_LONG
1229123971	0x4942f183	BAD_OPERATION_EXTRACT_USHORT
1229123972	0x4942f184	BAD_OPERATION_EXTRACT_ULONG
1229123973	0x4942f185	BAD_OPERATION_EXTRACT_FLOAT
1229123974	0x4942f186	BAD_OPERATION_EXTRACT_DOUBLE
1229123975	0x4942f187	BAD_OPERATION_EXTRACT_LONGLONG
1229123976	0x4942f188	BAD_OPERATION_EXTRACT_ULONGLONG
1229123977	0x4942f189	BAD_OPERATION_EXTRACT_BOOLEAN
1229123978	0x4942f18a	BAD_OPERATION_EXTRACT_CHAR
1229123979	0x4942f18b	BAD_OPERATION_EXTRACT_OCTET
1229123980	0x4942f18c	BAD_OPERATION_EXTRACT_WCHAR
1229123981	0x4942f18d	BAD_OPERATION_EXTRACT_STRING
1229123982	0x4942f18e	BAD_OPERATION_EXTRACT_WSTRING
1229123983	0x4942f18f	BAD_OPERATION_EXTRACT_ANY
1229123984	0x4942f190	BAD_OPERATION_INSERT_OBJECT_1
1229123985	0x4942f191	BAD_OPERATION_INSERT_OBJECT_2
1229123986	0x4942f192	BAD_OPERATION_EXTRACT_OBJECT_1
1229123987	0x4942f193	BAD_OPERATION_EXTRACT_OBJECT_2
1229123988	0x4942f194	BAD_OPERATION_EXTRACT_TYPECODE
1229123989	0x4942f195	BAD_OPERATION_EXTRACT_PRINCIPAL
1229123990	0x4942f196	BAD_OPERATION_EXTRACT_VALUE
1229123991	0x4942f197	BAD_OPERATION_GET_PRIMITIVE_TC_1
1229123992	0x4942f198	BAD_OPERATION_GET_PRIMITIVE_TC_2
1229123993	0x4942f199	BAD_OPERATION_INVOKE_NULL_PARAM_1
1229123994	0x4942f19a	BAD_OPERATION_INVOKE_NULL_PARAM_2
1229123995	0x4942f19b	BAD_OPERATION_INVOKE_DEFAULT_1
1229123996	0x4942f19c	BAD_OPERATION_INVOKE_DEFAULT_2
1229123997	0x4942f19d	BAD_OPERATION_UNKNOWN_BOOTSTRAP_METHOD
1229123998	0x4942f19e	BAD_OPERATION_EMPTY_ANY
1229123999	0x4942f19f	BAD_OPERATION_STUB_DISCONNECTED
1229124000	0x4942f1a0	BAD_OPERATION_TIE_DISCONNECTED
1229124001	0x4942f1a1	BAD_OPERATION_DELEGATE_DISCONNECTED
1229124097	0x4942f201	NULL_PARAM_1
1229124098	0x4942f202	NULL_PARAM_2
1229124099	0x4942f203	NULL_PARAM_3
1229124100	0x4942f204	NULL_PARAM_4
1229124101	0x4942f205	NULL_PARAM_5

Table 39. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229124102	0x4942f206	NULL_PARAM_6
1229124103	0x4942f207	NULL_PARAM_7
1229124104	0x4942f208	NULL_PARAM_8
1229124105	0x4942f209	NULL_PARAM_9
1229124106	0x4942f20a	NULL_PARAM_10
1229124107	0x4942f20b	NULL_PARAM_11
1229124108	0x4942f20c	NULL_PARAM_12
1229124109	0x4942f20d	NULL_PARAM_13
1229124110	0x4942f20e	NULL_PARAM_14
1229124111	0x4942f20f	NULL_PARAM_15
1229124112	0x4942f210	NULL_PARAM_16
1229124113	0x4942f211	NULL_PARAM_17
1229124114	0x4942f212	NULL_PARAM_18
1229124115	0x4942f213	NULL_PARAM_19
1229124116	0x4942f214	NULL_PARAM_20
1229124117	0x4942f215	NULL_IOR_OBJECT
1229124118	0x4942f216	NULL_PI_NAME
1229124119	0x4942f217	NULL_SC_DATA
1229124126	0x4942f21e	BAD_SERVANT_TYPE
1229124127	0x4942f21f	BAD_EXCEPTION
1229124128	0x4942f220	BAD_MODIFIER_LIST
1229124129	0x4942f221	NULL_PROP_MGR
1229124130	0x4942f222	INVALID_PROPERTY
1229124131	0x4942f223	ORBINITREF_FORMAT
1229124132	0x4942f224	ORBINITREF_MISSING_OBJECTURL
1229124133	0x4942f225	ORBDEFAULTINITREF_FORMAT
1229124134	0x4942f226	ORBDEFAULTINITREF_VALUE
1229124135	0x4942f227	OBJECTKEY_SERVERUUID_LENGTH
1229124136	0x4942f228	OBJECTKEY_SERVERUUID_NULL
1229124137	0x4942f229	BAD_REPOSITORY_ID
1229124138	0x4942f22a	BAD_PARAM_LOCAL_OBJECT
1229124139	0x4942f22b	NULL_OBJECT_IOR
1229124140	0x4942f22c	WRONG_ORB
1229124141	0x4942f22d	NULL_OBJECT_KEY
1229124142	0x4942f22e	NULL_OBJECT_URL
1229124143	0x4942f22f	NOT_A_NAMING_CONTEXT
1229124225	0x4942f281	TYPECODEIMPL_CTOR_MISUSE_1
1229124226	0x4942f282	TYPECODEIMPL_CTOR_MISUSE_2
1229124227	0x4942f283	TYPECODEIMPL_NULL_INDIRECTTYPE
1229124228	0x4942f284	TYPECODEIMPL_RECURSIVE_TYPECODES
1229124235	0x4942f28b	TYPECODEIMPL_KIND_INVALID_1
1229124236	0x4942f28c	TYPECODEIMPL_KIND_INVALID_2
1229124237	0x4942f28d	TYPECODEIMPL_NATIVE_1
1229124238	0x4942f28e	TYPECODEIMPL_NATIVE_2
1229124239	0x4942f28f	TYPECODEIMPL_NATIVE_3
1229124240	0x4942f290	TYPECODEIMPL_KIND_INDIRECT_1
1229124241	0x4942f291	TYPECODEIMPL_KIND_INDIRECT_2
1229124242	0x4942f292	TYPECODEIMPL_NULL_TYPECODE
1229124243	0x4942f293	TYPECODEIMPL_BODY_OF_TYPECODE
1229124244	0x4942f294	TYPECODEIMPL_KIND_RECURSIVE_1
1229124245	0x4942f295	TYPECODEIMPL_COMPLEX_DEFAULT_1

Table 39. Decimal minor exception codes 1229123841 to 1229124249 (continued)

1229124246	0x4942f296	TYPECODEIMPL_COMPLEX_DEFAULT_2
1229124247	0x4942f297	TYPECODEIMPL_INDIRECTION
1229124248	0x4942f298	TYPECODEIMPL_SIMPLE_DEFAULT
1229124249	0x4942f299	TYPECODEIMPL_NOT_CDROS

Table 40. Decimal minor exception codes 1229124250 to 1229125764

Decimal	Hexadecimal	Minor code reason
1229124250	0x4942f29a	TYPECODEIMPL_NO_IMPLEMENT_1
1229124251	0x4942f29b	TYPECODEIMPL_NO_IMPLEMENT_2
1229124357	0x4942f305	CONN_PURGE_REBIND
1229124358	0x4942f306	CONN_PURGE_ABORT
1229124359	0x4942f307	CONN_NOT_ESTABLISH
1229124360	0x4942f308	CONN_CLOSE_REBIND
1229124368	0x4942f310	WRITE_ERROR_SEND
1229124376	0x4942f318	GET_PROPERTIES_ERROR
1229124384	0x4942f320	BOOTSTRAP_SERVER_NOT_AVAIL
1229124392	0x4942f328	INVOKE_ERROR
1229124481	0x4942f381	BAD_HEX_DIGIT
1229124482	0x4942f382	BAD_STRINGIFIED_IOR_LEN
1229124483	0x4942f383	BAD_STRINGIFIED_IOR
1229124485	0x4942f385	BAD_MODIFIER_1
1229124486	0x4942f386	BAD_MODIFIER_2
1229124488	0x4942f388	CODESET_INCOMPATIBLE
1229124490	0x4942f38a	LONG_DOUBLE_NOT_IMPLEMENTED_1
1229124491	0x4942f38b	LONG_DOUBLE_NOT_IMPLEMENTED_2
1229124492	0x4942f38c	LONG_DOUBLE_NOT_IMPLEMENTED_3
1229124496	0x4942f390	COMPLEX_TYPES_NOT_IMPLEMENTED
1229124497	0x4942f391	VALUE_BOX_NOT_IMPLEMENTED
1229124498	0x4942f392	NULL_STRINGIFIED_IOR
1229124865	0x4942f501	TRANS_NS_CANNOT_CREATE_INITIAL_NC_SYS
1229124866	0x4942f502	TRANS_NS_CANNOT_CREATE_INITIAL_NC
1229124867	0x4942f503	GLOBAL_ORB_EXISTS
1229124868	0x4942f504	PLUGINS_ERROR
1229124869	0x4942f505	INCOMPATIBLE_JDK_VERSION
1229124993	0x4942f581	BAD_REPLYSTATUS
1229124994	0x4942f582	PEEKSTRING_FAILED
1229124995	0x4942f583	GET_LOCAL_HOST_FAILED
1229124996	0x4942f584	CREATE_LISTENER_FAILED
1229124997	0x4942f585	BAD_LOCATE_REQUEST_STATUS
1229124998	0x4942f586	STRINGIFY_WRITE_ERROR
1229125000	0x4942f588	BAD_GIOP_REQUEST_TYPE_1
1229125001	0x4942f589	BAD_GIOP_REQUEST_TYPE_2
1229125002	0x4942f58a	BAD_GIOP_REQUEST_TYPE_3
1229125003	0x4942f58b	BAD_GIOP_REQUEST_TYPE_4
1229125005	0x4942f58d	NULL_ORB_REFERENCE
1229125006	0x4942f58e	NULL_NAME_REFERENCE
1229125008	0x4942f590	ERROR_UNMARSHALING_USEREXC
1229125009	0x4942f591	SUBCONTRACTREGISTRY_ERROR
1229125010	0x4942f592	LOCATIONFORWARD_ERROR
1229125011	0x4942f593	BAD_READER_THREAD
1229125013	0x4942f595	BAD_REQUEST_ID

Table 40. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229125014	0x4942f596	BAD_SYSTEMEXCEPTION
1229125015	0x4942f597	BAD_COMPLETION_STATUS
1229125016	0x4942f598	INITIAL_REF_ERROR
1229125017	0x4942f599	NO_CODEC_FACTORY
1229125018	0x4942f59a	BAD_SUBCONTRACT_ID
1229125019	0x4942f59b	BAD_SYSTEMEXCEPTION_2
1229125020	0x4942f59c	NOT_PRIMITIVE_TYPECODE
1229125021	0x4942f59d	BAD_SUBCONTRACT_ID_2
1229125022	0x4942f59e	BAD_SUBCONTRACT_TYPE
1229125023	0x4942f59f	NAMING_CTX_REBIND_ALREADY_BOUND
1229125024	0x4942f5a0	NAMING_CTX_REBINDCTX_ALREADY_BOUND
1229125025	0x4942f5a1	NAMING_CTX_BAD_BINDINGTYPE
1229125026	0x4942f5a2	NAMING_CTX_RESOLVE_CANNOT_NARROW_TO_CTX
1229125032	0x4942f5a8	TRANS_NC_BIND_ALREADY_BOUND
1229125033	0x4942f5a9	TRANS_NC_LIST_GOT_EXC
1229125034	0x4942f5aa	TRANS_NC_NEWCTX_GOT_EXC
1229125035	0x4942f5ab	TRANS_NC_DESTROY_GOT_EXC
1229125036	0x4942f5ac	TRANS_BI_DESTROY_GOT_EXC
1229125042	0x4942f5b2	INVALID_CHAR_CODESET_1
1229125043	0x4942f5b3	INVALID_CHAR_CODESET_2
1229125044	0x4942f5b4	INVALID_WCHAR_CODESET_1
1229125045	0x4942f5b5	INVALID_WCHAR_CODESET_2
1229125046	0x4942f5b6	GET_HOST_ADDR_FAILED
1229125047	0x4942f5b7	REACHED_UNREACHABLE_PATH
1229125048	0x4942f5b8	PROFILE_CLONE_FAILED
1229125049	0x4942f5b9	INVALID_LOCATE_REQUEST_STATUS
1229125050	0x4942f5ba	NO_UNSAFE_CLASS
1229125051	0x4942f5bb	REQUEST_WITHOUT_CONNECTION_1
1229125052	0x4942f5bc	REQUEST_WITHOUT_CONNECTION_2
1229125053	0x4942f5bd	REQUEST_WITHOUT_CONNECTION_3
1229125054	0x4942f5be	REQUEST_WITHOUT_CONNECTION_4
1229125055	0x4942f5bf	REQUEST_WITHOUT_CONNECTION_5
1229125056	0x4942f5c0	NOT_USED_1
1229125057	0x4942f5c1	NOT_USED_2
1229125058	0x4942f5c2	NOT_USED_3
1229125059	0x4942f5c3	NOT_USED_4
1229125512	0x4942f788	BAD_CODE_SET
1229125520	0x4942f790	INV_RMI_STUB
1229125521	0x4942f791	INV_LOAD_STUB
1229125522	0x4942f792	INV_OBJ_IMPLEMENTATION
1229125523	0x4942f793	OBJECTKEY_NOMAGIC
1229125524	0x4942f794	OBJECTKEY_NOSCID
1229125525	0x4942f795	OBJECTKEY_NOSERVERID
1229125526	0x4942f796	OBJECTKEY_NOSERVERUUID
1229125527	0x4942f797	OBJECTKEY_SERVERUUIDKEY
1229125528	0x4942f798	OBJECTKEY_NOPOANAME
1229125529	0x4942f799	OBJECTKEY_SETSCID
1229125530	0x4942f79a	OBJECTKEY_SETSERVERID
1229125531	0x4942f79b	OBJECTKEY_NOUSERKEY
1229125532	0x4942f79c	OBJECTKEY_SETUSERKEY
1229125533	0x4942f79d	INVALID_INDEXED_PROFILE_1

Table 40. Decimal minor exception codes 1229124250 to 1229125764 (continued)

1229125534	0x4942f79e	INVALID_INDEXED_PROFILE_2
1229125762	0x4942f882	UNSPECIFIED_MARSHAL_1
1229125763	0x4942f883	UNSPECIFIED_MARSHAL_2
1229125764	0x4942f884	UNSPECIFIED_MARSHAL_3

Table 41. Decimal minor exception codes 1299125765 to 1229125906

Decimal	Hexadecimal	Minor code reason
1229125765	0x4942f885	UNSPECIFIED_MARSHAL_4
1229125766	0x4942f886	UNSPECIFIED_MARSHAL_5
1229125767	0x4942f887	UNSPECIFIED_MARSHAL_6
1229125768	0x4942f888	UNSPECIFIED_MARSHAL_7
1229125769	0x4942f889	UNSPECIFIED_MARSHAL_8
1229125770	0x4942f88a	UNSPECIFIED_MARSHAL_9
1229125771	0x4942f88b	UNSPECIFIED_MARSHAL_10
1229125772	0x4942f88c	UNSPECIFIED_MARSHAL_11
1229125773	0x4942f88d	UNSPECIFIED_MARSHAL_12
1229125774	0x4942f88e	UNSPECIFIED_MARSHAL_13
1229125775	0x4942f88f	UNSPECIFIED_MARSHAL_14
1229125776	0x4942f890	UNSPECIFIED_MARSHAL_15
1229125777	0x4942f891	UNSPECIFIED_MARSHAL_16
1229125778	0x4942f892	UNSPECIFIED_MARSHAL_17
1229125779	0x4942f893	UNSPECIFIED_MARSHAL_18
1229125780	0x4942f894	UNSPECIFIED_MARSHAL_19
1229125781	0x4942f895	UNSPECIFIED_MARSHAL_20
1229125782	0x4942f896	UNSPECIFIED_MARSHAL_21
1229125783	0x4942f897	UNSPECIFIED_MARSHAL_22
1229125784	0x4942f898	UNSPECIFIED_MARSHAL_23
1229125785	0x4942f899	UNSPECIFIED_MARSHAL_24
1229125786	0x4942f89a	UNSPECIFIED_MARSHAL_25
1229125787	0x4942f89b	UNSPECIFIED_MARSHAL_26
1229125788	0x4942f89c	UNSPECIFIED_MARSHAL_27
1229125789	0x4942f89d	UNSPECIFIED_MARSHAL_28
1229125790	0x4942f89e	UNSPECIFIED_MARSHAL_29
1229125791	0x4942f89f	UNSPECIFIED_MARSHAL_30
1229125792	0x4942f8a0	UNSPECIFIED_MARSHAL_31
1229125793	0x4942f8a1	UNSPECIFIED_MARSHAL_32
1229125794	0x4942f8a2	UNSPECIFIED_MARSHAL_33
1229125795	0x4942f8a3	UNSPECIFIED_MARSHAL_34
1229125796	0x4942f8a4	UNSPECIFIED_MARSHAL_35
1229125797	0x4942f8a5	UNSPECIFIED_MARSHAL_36
1229125798	0x4942f8a6	UNSPECIFIED_MARSHAL_37
1229125799	0x4942f8a7	UNSPECIFIED_MARSHAL_38
1229125800	0x4942f8a8	UNSPECIFIED_MARSHAL_39
1229125801	0x4942f8a9	UNSPECIFIED_MARSHAL_40
1229125802	0x4942f8aa	UNSPECIFIED_MARSHAL_41
1229125803	0x4942f8ab	UNSPECIFIED_MARSHAL_42
1229125804	0x4942f8ac	UNSPECIFIED_MARSHAL_43
1229125805	0x4942f8ad	UNSPECIFIED_MARSHAL_44
1229125806	0x4942f8ae	UNSPECIFIED_MARSHAL_45
1229125807	0x4942f8af	UNSPECIFIED_MARSHAL_46
1229125808	0x4942f8b0	UNSPECIFIED_MARSHAL_47

Table 41. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125809	0x4942f8b1	UNSPECIFIED_MARSHAL_48
1229125810	0x4942f8b2	UNSPECIFIED_MARSHAL_49
1229125811	0x4942f8b3	UNSPECIFIED_MARSHAL_50
1229125812	0x4942f8b4	UNSPECIFIED_MARSHAL_51
1229125813	0x4942f8b5	UNSPECIFIED_MARSHAL_52
1229125814	0x4942f8b6	UNSPECIFIED_MARSHAL_53
1229125815	0x4942f8b7	UNSPECIFIED_MARSHAL_54
1229125816	0x4942f8b8	UNSPECIFIED_MARSHAL_55
1229125817	0x4942f8b9	UNSPECIFIED_MARSHAL_56
1229125818	0x4942f8ba	UNSPECIFIED_MARSHAL_57
1229125819	0x4942f8bb	UNSPECIFIED_MARSHAL_58
1229125820	0x4942f8bc	UNSPECIFIED_MARSHAL_59
1229125821	0x4942f8bd	UNSPECIFIED_MARSHAL_60
1229125822	0x4942f8be	UNSPECIFIED_MARSHAL_61
1229125823	0x4942f8bf	UNSPECIFIED_MARSHAL_62
1229125824	0x4942f8c0	UNSPECIFIED_MARSHAL_63
1229125825	0x4942f8c1	UNSPECIFIED_MARSHAL_64
1229125826	0x4942f8c2	UNSPECIFIED_MARSHAL_65
1229125827	0x4942f8c3	UNSPECIFIED_MARSHAL_66
1229125828	0x4942f8c4	READ_OBJECT_EXCEPTION_2
1229125841	0x4942f8d1	UNSUPPORTED_IDLTYPE
1229125842	0x4942f8d2	DSI_RESULT_EXCEPTION
1229125844	0x4942f8d4	IIOPINPUTSTREAM_GROW
1229125847	0x4942f8d7	NO_CHAR_CONVERTER_1
1229125848	0x4942f8d8	NO_CHAR_CONVERTER_2
1229125849	0x4942f8d9	CHARACTER_MALFORMED_1
1229125850	0x4942f8da	CHARACTER_MALFORMED_2
1229125851	0x4942f8db	CHARACTER_MALFORMED_3
1229125852	0x4942f8dc	CHARACTER_MALFORMED_4
1229125854	0x4942f8de	INCORRECT_CHUNK_LENGTH
1229125856	0x4942f8e0	CHUNK_OVERFLOW
1229125858	0x4942f8e2	CANNOT_GROW
1229125859	0x4942f8e3	CODESET_ALREADY_SET
1229125860	0x4942f8e4	REQUEST_CANCELLED
1229125861	0x4942f8e5	WRITE_TO_STREAM_1
1229125862	0x4942f8e6	WRITE_TO_STREAM_2
1229125863	0x4942f8e7	WRITE_TO_STREAM_3
1229125864	0x4942f8e8	WRITE_TO_STREAM_4
1229125865	0x4942f8e9	PROXY_MARSHAL_FAILURE
1229125866	0x4942f8ea	PROXY_UNMARSHAL_FAILURE
1229125867	0x4942f8eb	INVALID_START_VALUE
1229125868	0x4942f8ec	INVALID_CHUNK_STATE
1229125869	0x4942f8ed	NULL_CODEBASE_IOR
1229125889	0x4942f901	DSI_NOT_IMPLEMENTED
1229125890	0x4942f902	GETINTERFACE_NOT_IMPLEMENTED
1229125891	0x4942f903	SEND_DEFERRED_NOTIMPLEMENTED
1229125893	0x4942f905	ARGUMENTS_NOTIMPLEMENTED
1229125894	0x4942f906	RESULT_NOTIMPLEMENTED
1229125895	0x4942f907	EXCEPTIONS_NOTIMPLEMENTED
1229125896	0x4942f908	CONTEXTLIST_NOTIMPLEMENTED
1229125902	0x4942f90e	CREATE_OBJ_REF_BYTE_NOTIMPLEMENTED

Table 41. Decimal minor exception codes 1299125765 to 1229125906 (continued)

1229125903	0x4942f90f	CREATE_OBJ_REF_IOR_NOTIMPLEMENTED
1229125904	0x4942f910	GET_KEY_NOTIMPLEMENTED
1229125905	0x4942f911	GET_IMPL_ID_NOTIMPLEMENTED
1229125906	0x4942f912	GET_SERVANT_NOTIMPLEMENTED

Table 42. Decimal minor exception codes 1229125907 to 1229126567

Decimal	Hexadecimal	Minor code reason
1229125907	0x4942f913	SET_ORB_NOTIMPLEMENTED
1229125908	0x4942f914	SET_ID_NOTIMPLEMENTED
1229125909	0x4942f915	GET_CLIENT_SUBCONTRACT_NOTIMPLEMENTED
1229125913	0x4942f919	CONTEXTIMPL_NOTIMPLEMENTED
1229125914	0x4942f91a	CONTEXT_NAME_NOTIMPLEMENTED
1229125915	0x4942f91b	PARENT_NOTIMPLEMENTED
1229125916	0x4942f91c	CREATE_CHILD_NOTIMPLEMENTED
1229125917	0x4942f91d	SET_ONE_VALUE_NOTIMPLEMENTED
1229125918	0x4942f91e	SET_VALUES_NOTIMPLEMENTED
1229125919	0x4942f91f	DELETE_VALUES_NOTIMPLEMENTED
1229125920	0x4942f920	GET_VALUES_NOTIMPLEMENTED
1229125922	0x4942f922	GET_CURRENT_NOTIMPLEMENTED_1
1229125923	0x4942f923	GET_CURRENT_NOTIMPLEMENTED_2
1229125924	0x4942f924	CREATE_OPERATION_LIST_NOTIMPLEMENTED_1
1229125925	0x4942f925	CREATE_OPERATION_LIST_NOTIMPLEMENTED_2
1229125926	0x4942f926	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_1
1229125927	0x4942f927	GET_DEFAULT_CONTEXT_NOTIMPLEMENTED_2
1229125928	0x4942f928	SHUTDOWN_NOTIMPLEMENTED
1229125929	0x4942f929	WORK_PENDING_NOTIMPLEMENTED
1229125930	0x4942f92a	PERFORM_WORK_NOTIMPLEMENTED
1229125931	0x4942f92b	COPY_TK_ABSTRACT_NOTIMPLEMENTED
1229125932	0x4942f92c	PI_CLIENT_GET_POLICY_NOTIMPLEMENTED
1229125933	0x4942f92d	PI_SERVER_GET_POLICY_NOTIMPLEMENTED
1229125934	0x4942f92e	ADDRESSING_MODE_NOTIMPLEMENTED_1
1229125935	0x4942f92f	ADDRESSING_MODE_NOTIMPLEMENTED_2
1229125936	0x4942f930	SET_OBJECT_RESOLVER_NOTIMPLEMENTED
1229125937	0x4942f931	DISCONNECTED_SERVANT_1
1229125938	0x4942f932	DISCONNECTED_SERVANT_2
1229125939	0x4942f933	DISCONNECTED_SERVANT_3
1229125940	0x4942f934	DISCONNECTED_SERVANT_4
1229125941	0x4942f935	DISCONNECTED_SERVANT_5
1229125942	0x4942f936	DISCONNECTED_SERVANT_6
1229125943	0x4942f937	DISCONNECTED_SERVANT_7
1229125944	0x4942f938	GET_INTERFACE_DEF_NOT_IMPLEMENTED
1229126017	0x4942f981	MARSHAL_NO_MEMORY_1
1229126018	0x4942f982	MARSHAL_NO_MEMORY_2
1229126019	0x4942f983	MARSHAL_NO_MEMORY_3
1229126020	0x4942f984	MARSHAL_NO_MEMORY_4
1229126021	0x4942f985	MARSHAL_NO_MEMORY_5
1229126022	0x4942f986	MARSHAL_NO_MEMORY_6
1229126023	0x4942f987	MARSHAL_NO_MEMORY_7
1229126024	0x4942f988	MARSHAL_NO_MEMORY_8
1229126025	0x4942f989	MARSHAL_NO_MEMORY_9
1229126026	0x4942f98a	MARSHAL_NO_MEMORY_10

Table 42. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229126027	0x4942f98b	MARSHAL_NO_MEMORY_11
1229126028	0x4942f98c	MARSHAL_NO_MEMORY_12
1229126029	0x4942f98d	MARSHAL_NO_MEMORY_13
1229126030	0x4942f98e	MARSHAL_NO_MEMORY_14
1229126031	0x4942f98f	MARSHAL_NO_MEMORY_15
1229126032	0x4942f990	MARSHAL_NO_MEMORY_16
1229126033	0x4942f991	MARSHAL_NO_MEMORY_17
1229126034	0x4942f992	MARSHAL_NO_MEMORY_18
1229126035	0x4942f993	MARSHAL_NO_MEMORY_19
1229126036	0x4942f994	MARSHAL_NO_MEMORY_20
1229126037	0x4942f995	MARSHAL_NO_MEMORY_21
1229126038	0x4942f996	MARSHAL_NO_MEMORY_22
1229126039	0x4942f997	MARSHAL_NO_MEMORY_23
1229126040	0x4942f998	MARSHAL_NO_MEMORY_24
1229126041	0x4942f999	MARSHAL_NO_MEMORY_25
1229126042	0x4942f99a	MARSHAL_NO_MEMORY_26
1229126043	0x4942f99b	MARSHAL_NO_MEMORY_27
1229126044	0x4942f99c	MARSHAL_NO_MEMORY_28
1229126045	0x4942f99d	MARSHAL_NO_MEMORY_29
1229126046	0x4942f99e	MARSHAL_NO_MEMORY_30
1229126047	0x4942f99f	MARSHAL_NO_MEMORY_31
1229126401	0x4942fb01	RESPONSE_TIMED_OUT
1229126402	0x4942fb02	FRAGMENT_TIMED_OUT
1229126529	0x4942fb81	NO_SERVER_SC_IN_DISPATCH
1229126530	0x4942fb82	NO_SERVER_SC_IN_LOOKUP
1229126531	0x4942fb83	NO_SERVER_SC_IN_CREATE_DEFAULT_SERVER
1229126532	0x4942fb84	NO_SERVER_SC_IN_SETUP
1229126533	0x4942fb85	NO_SERVER_SC_IN_LOCATE
1229126534	0x4942fb86	NO_SERVER_SC_IN_DISCONNECT
1229126539	0x4942fb8b	ORB_CONNECT_ERROR_1
1229126540	0x4942fb8c	ORB_CONNECT_ERROR_2
1229126541	0x4942fb8d	ORB_CONNECT_ERROR_3
1229126542	0x4942fb8e	ORB_CONNECT_ERROR_4
1229126543	0x4942fb8f	ORB_CONNECT_ERROR_5
1229126544	0x4942fb90	ORB_CONNECT_ERROR_6
1229126545	0x4942fb91	ORB_CONNECT_ERROR_7
1229126546	0x4942fb92	ORB_CONNECT_ERROR_8
1229126547	0x4942fb93	ORB_CONNECT_ERROR_9
1229126548	0x4942fb94	ORB_REGISTER_1
1229126549	0x4942fb95	ORB_REGISTER_2
1229126553	0x4942fb99	ORB_REGISTER_LOCAL_1
1229126554	0x4942fb9a	ORB_REGISTER_LOCAL_2
1229126555	0x4942fb9b	LOCAL_SERVANT_LOOKUP
1229126556	0x4942fb9c	POA_LOOKUP_ERROR
1229126557	0x4942fb9d	POA_INACTIVE
1229126558	0x4942fb9e	POA_NO_SERVANT_MANAGER
1229126559	0x4942fb9f	POA_NO_DEFAULT_SERVANT
1229126560	0x4942fba0	POA_WRONG_POLICY
1229126561	0x4942fba1	FINDPOA_ERROR
1229126562	0x4942fba2	ADAPTER_ACTIVATOR_EXCEPTION
1229126563	0x4942fba3	POA_SERVANT_ACTIVATOR_LOOKUP_FAILED

Table 42. Decimal minor exception codes 1229125907 to 1229126567 (continued)

1229126564	0x4942fba4	POA_BAD_SERVANT_MANAGER
1229126565	0x4942fba5	POA_SERVANT_LOCATOR_LOOKUP_FAILED
1229126566	0x4942fba6	POA_UNKNOWN_POLICY
1229126567	0x4942fba7	POA_NOT_FOUND

Table 43. Decimal minor exception codes 1229126568 to 1330446377

Decimal	Hexadecimal	Minor code reason
1229126568	0x4942fba8	SERVANT_LOOKUP
1229126569	0x4942fba9	SERVANT_IS_ACTIVE
1229126570	0x4942fbaa	SERVANT_DISPATCH
1229126571	0x4942fbab	WRONG_CLIENTSC
1229126572	0x4942fbac	WRONG_SERVERSC
1229126573	0x4942fbad	SERVANT_IS_NOT_ACTIVE
1229126574	0x4942fbae	POA_WRONG_POLICY_1
1229126575	0x4942fbaf	POA_NOCONTEXT_1
1229126576	0x4942fbb0	POA_NOCONTEXT_2
1229126577	0x4942fbb1	POA_INVALID_NAME_1
1229126578	0x4942fbb2	POA_INVALID_NAME_2
1229126579	0x4942fbb3	POA_INVALID_NAME_3
1229126580	0x4942fbb4	POA_NOCONTEXT_FOR_PREINVOKE
1229126581	0x4942fbb5	POA_NOCONTEXT_FOR_CHECKING_PREINVOKE
1229126582	0x4942fbb6	POA_NOCONTEXT_FOR_POSTINVOKE
1229126657	0x4942fc01	LOCATE_UNKNOWN_OBJECT
1229126658	0x4942fc02	BAD_SERVER_ID_1
1229126659	0x4942fc03	BAD_SERVER_ID_2
1229126660	0x4942fc04	BAD_IMPLID
1229126665	0x4942fc09	BAD_SKELETON_1
1229126666	0x4942fc0a	BAD_SKELETON_2
1229126673	0x4942fc11	SERVANT_NOT_FOUND_1
1229126674	0x4942fc12	SERVANT_NOT_FOUND_2
1229126675	0x4942fc13	SERVANT_NOT_FOUND_3
1229126676	0x4942fc14	SERVANT_NOT_FOUND_4
1229126677	0x4942fc15	SERVANT_NOT_FOUND_5
1229126678	0x4942fc16	SERVANT_NOT_FOUND_6
1229126679	0x4942fc17	SERVANT_NOT_FOUND_7
1229126687	0x4942fc1f	SERVANT_DISCONNECTED_1
1229126688	0x4942fc20	SERVANT_DISCONNECTED_2
1229126689	0x4942fc21	NULL_SERVANT
1229126690	0x4942fc22	ADAPTER_ACTIVATOR_FAILED
1229126692	0x4942fc24	ORB_DESTROYED
1229126693	0x4942fc25	DYNANY_DESTROYED
1229127170	0x4942fe02	CONNECT_FAILURE_1
1229127171	0x4942fe03	CONNECT_FAILURE_2
1229127172	0x4942fe04	CONNECT_FAILURE_3
1229127173	0x4942fe05	CONNECT_FAILURE_4
1229127297	0x4942fe81	UNKNOWN_CORBA_EXC
1229127298	0x4942fe82	RUNTIMEEXCEPTION
1229127299	0x4942fe83	UNKNOWN_SERVER_ERROR
1229127300	0x4942fe84	UNKNOWN_DSI_SYSEX
1229127301	0x4942fe85	UNEXPECTED_CHECKED_EXCEPTION
1229127302	0x4942fe86	UNKNOWN_CREATE_EXCEPTION_RESPONSE

Table 43. Decimal minor exception codes 1229126568 to 1330446377 (continued)

1229127312	0x4942fe90	UNKNOWN_PI_EXC
1229127313	0x4942fe91	UNKNOWN_PI_EXC_2
1229127314	0x4942fe92	PI_ARGS_FAILURE
1229127315	0x4942fe93	PI_EXCEPTS_FAILURE
1229127316	0x4942fe94	PI_CONTEXTS_FAILURE
1229127317	0x4942fe95	PI_OP_CONTEXT_FAILURE
1229127326	0x4942fe9e	USER_DEFINED_ERROR
1229127327	0x4942fe9f	UNKNOWN_RUNTIME_IN_BOOTSTRAP
1229127328	0x4942fea0	UNKNOWN_THROWABLE_IN_BOOTSTRAP
1229127329	0x4942fea1	UNKNOWN_RUNTIME_IN_INSAGENT
1229127330	0x4942fea2	UNKNOWN_THROWABLE_IN_INSAGENT
1229127331	0x4942fea3	UNEXPECTED_IN_PROCESSING_CLIENTSIDE_INTERCEPTOR
1229127332	0x4942fea4	UNEXPECTED_IN_PROCESSING_SERVERSIDE_INTERCEPTOR
1229127333	0x4942fea5	UNEXPECTED_PI_LOCAL_REQUEST
1229127334	0x4942fea6	UNEXPECTED_PI_LOCAL_RESPONSE
1330446336	0x4f4d0000	OMGVMCID
1330446337	0x4f4d0001	FAILURE_TO_REGISTER_OR_LOOKUP_VALUE_FACTORY
1330446338	0x4f4d0002	RID_ALREADY_DEFINED_IN_IFR
1330446339	0x4f4d0003	IN_INVOCATION_CONTEXT
1330446340	0x4f4d0004	ORB_SHUTDOWN
1330446341	0x4f4d0005	NAME_CLASH_IN_INHERITED_CONTEXT
1330446342	0x4f4d0006	SERVANT_MANAGER_EXISTS
1330446343	0x4f4d0007	INS_BAD_SCHEME_NAME
1330446344	0x4f4d0008	INS_BAD_ADDRESS
1330446345	0x4f4d0009	INS_BAD_SCHEME_SPECIFIC_PART
1330446346	0x4f4d000a	INS_OTHER
1330446348	0x4f4d000c	POLICY_FACTORY_EXISTS
1330446350	0x4f4d000e	INVALID_PI_CALL
1330446351	0x4f4d000f	SERVICE_CONTEXT_ID_EXISTS
1330446353	0x4f4d0011	POA_DESTROYED
1330446359	0x4f4d0017	NO_TRANSMISSION_CODE
1330446362	0x4f4d001a	INVALID_SERVICE_CONTEXT
1330446363	0x4f4d001b	NULL_OBJECT_ON_REGISTER
1330446364	0x4f4d001c	INVALID_COMPONENT_ID
1330446365	0x4f4d001d	INVALID_IOR_PROFILE
1330446375	0x4f4d0027	INVALID_STREAM_FORMAT_1
1330446376	0x4f4d0028	NOT_VALUE_OUTPUT_STREAM
1330446377	0x4f4d0029	NOT_VALUE_INPUT_STREAM

If none of these steps fixes your problem, check to see if the problem has been identified and documented by looking at the available online support (hints and tips, technotes, and fixes).

For current information available from IBM Support on known problems and their resolution, see the i5/OS software page. You should also refer to this page before opening a PMR because it contains information about the documents that you have to gather and send to IBM to receive help with a problem.

Enabling HTTP tunneling

HTTP tunneling enables clients, that reside outside of a firewall, to bundle all of the information, that the client-side Object Request Broker (ORB) needs to send to the server-side ORB, into a normal HTTP request. This request can then be sent to the server on port 80, just like any other HTTP request.

Before you begin

Make sure that the client-side ORB is an IBM ORB. Tunneling does not work if you are using a non-IBM ORB on the client.

Also, if Secure Sockets Layer (SSL) security is required for the tunneling, make sure that the required certificates and key files are configured.

About this task

Sometimes clients residing outside of a firewall need to communicate with modules, such as EJB modules, that reside on a server inside of the firewall. The client-side and server-side ORBs manage this interaction between the client and the server. However, firewalls normally block the ports that a client, uses to talk to the server-side ORB. Therefore if your installation uses a firewall that blocks the ports a client uses to talk to the server-side ORB, you should set up HTTP tunneling.

The IIOPTunnelServlet, which is shipped with the product as class file com.ibm.CORBA.services.IIOPTunnelServlet.class, allows an HTTP client, such as a Java client, that is embedded with RMI-IIOP, to communicate with a server that resides inside of a firewall. This class file, along with the following three class files, are bundled within the WAS_HOME/plugins/com.ibm.ws.runtime_6.1.0.jar file. These additional class files enhance the servlet's capabilities.

com.ibm.CORBA.services.redirector.ConnectionStream.class
com.ibm.CORBA.services.redirector.Redirector.class
com.ibm.CORBA.services.redirector.RedirectorController.class

When tunneling is enabled, the IIOPTunnelServlet servlet on the server receives the HTTP request and unpacks all of the ORB information. The servlet then calls the server-side ORB on the client's behalf. The server-side ORB treats the request as it would treat any normal ORB request and responds to the servlet. The servlet packs the ORB response into an HTTP response and sends the response back to the client-side ORB, through the firewall. The client-side ORB unpacks the HTTP response and pulls out the response.

Tunneling can operate over HTTPS as well as over HTTP. Therefore, you can use Secure Sockets Layer (SSL) security to secure your tunneling clients if your security procedures require that all communication to your servers is SSL secured.

1. Create an installable IIOPTunnel.ear file that includes the IIOPTunnelServlet servlet.

   Before you can run the IIOPTunnelServlet servlet on the server, you must make it part of an application that you can install on the server. You can use an application assembly tool to create an installable IIOPTunnel.ear file that includes this servlet. For example, if you use the assembly tool that is shipped with the product:

   a. Start the tool.
   b. Open the WEB perspective.
   c. In the Project Explorer view, right click in an empty pane and select **New > Dynamic Web Project**.
   d. In the Create Dynamic Web Project wizard, change the project Name to IIOPTunnel, or another name that is meaningful to you. By default, the **Add Module to an EAR project** option is selected, the EAR project name is set to IIOPTunnelEAR, and the Context Root is set to IIOPTunnel.
   e. Keep these default settings and click **Finish**.
   f. Add the com.ibm.ws.runtime_7.0.0.jar file to the Web Project Build Path.

      Before you can register the new servlet in the Web Deployment Descriptor, you must add the IIOPTunnelServlet servlet, that resides in the WAS_HOME/lib/plugins/com.ibm.ws.runtime_7.0.0.jar file, to your build path.

      1) Right click the IIOPTunnel Web Project, and select **Properties > Java Build Path**.
      2) Select the Libraries tab and press the Add external JARs button.

3) Add the com.ibm.ws.runtime_7.0.0.jar file, and then click **OK**.

2. Export your EAR file.
   a. Right click on the IIOPTunnelEAR project.
   b. Click **Export > EAR File**, browse to your selected destination directory and specify the EAR file name as IIOPTunnel.ear, or the file name that you specified in Step 1d.
   c. Click **Finish**.

   You get your IIOPTunnel.ear file, which is ready for you to deploy.
3. Install the IIOPTunnel.ear file on your target application server. You can accept all default values during installation.

   Remember to adjust the tunnelAgentURL in the client to reflect the actual location of the IIOPTunnelServlet on your server.

   Note:

   `http(s)://host_name:port/context_root/Servlet_URLmapping`

   The*host_name:port* are the host name and port that is assigned to teh server on which the IIOPTunnelServlet resides. The port can be either an HTTP or an HTTPS port, depending on your security requirements.

   The *context_root* and *Servlet_URLmapping* values must match the values that are defined for the context-root and servlet-URLmapping elements in the servlet web.xml file.

   For example, if the servlet is installed on the default server, and context-root=iioptunnel, and Servlet-URLmapping=tunnel, the following URL must be specified for tunnelAgentURL in the client:

   `http://localhost:9080/IIOPTunnel/IIOPTunnelServlet`

   To verify that the servlet is deployed and running successfully, you can open a browser and point to http:// hostname:9080/iioptunnel/tunnel. If the servlet is working, the browser tries to download the servlet as if it were just a normal file. You can then cancel the download.
4. Verify that the servlet is deployed and running successfully

   To verify that the servlet is deployed and running successfully, you can open a browser and point to http:// hostname:9080/IIOPTunnel/IIOPTunnelServlet. If the servlet is working, the browser tries o download the servlet as if it were just a normal file. Simply cancel the download.

   Specify the following parameters if you encounter a problem deploying and running the servlet.

   `-Dcom.ibm.CORBA.TunnelAgentURL=https://localhost:9080/IIOPTunnel/IIOPTunnelServlet?debug=true`
5. Configure the ORB Service for the client-side ORB to enable tunneling

   The client determines whether standard IIOP and HTTP tunneling should be used for communication with the server-side ORB. Therefore you must set the following ORB properties on the client.

   ```
   com.ibm.CORBA.ForceTunnel=ALWAYS
   com.ibm.CORBA.TunnelAgentURL=http://host_name:9080/IIOPTunnel/IIOPTunnelServlet com.ibm.CORBA.FragmentSize=0
   ```

   To enabled tunneling on the client ORB, the com.ibm.CORBA.ForceTunnel property must be set to **ALWAYS**. This setting indicates that this client is always going to tunnel. Other values that can be specified for the com.ibm.CORBA.ForceTunnel property are:

   NEVER, which indicates that you want to disable HTTP tunneling. If a TCP connection fails, a CORBA system exception (COMM_FAILURE) occurs.

   WHENREQUIRED, which indicates that you want to use HTTP tunneling if TCP connections fail.

   The second property specifies the fully qualified URL at which the tunneling servlet is reached. The port 9080 is the WC_defaulthost port for the server. The port number you specify must match the port number that is specified in the configuration file, serverindex.xml, for the server on which the IIOPTunnelServlet servlet resides.

   The third property turns off ORB fragmenting. Normally, the ORB breaks up communications into fragments, to improve performance, but tunneling will not work if the ORB is fragmenting.

   You can also set these properties by adding them as parameters to the JVM command line:

```
-Dcom.ibm.CORBA.ForceTunnel=always
-Dcom.ibm.CORBA.TunnelAgentURL=http://host_name:9080/iioptunnel/tunnel
-Dcom.ibm.CORBA.FragmentSize=0
```

Optionally, you can also set the following property if you want to specify client-side security settings:

```
-Dcom.ibm.CORBA.ConfigURL=file:PROFILE_ROOT/properties/sas.client.props
```

6. Turn off fragmenting on the server-side ORB. The only property that you must configure for the server-side ORB to enable tunneling is the com.ibm.CORBA.FragmentSize property. This property must be set to `0` to turn off fragmenting.
   a. In the administrative console, click **Servers > Server Types > WebSphere application servers**, and click the server where the tunneling servlet is installed.
   b. Click **ORB Service**, then click **Custom properties**.
   c. Click **New** and then specify **com.ibm.CORBA.FragmentSize** in the Name field and **0** in the Value field.
   d. Click **OK**, and then save the changes.
7. Stop and then restart the application server.

What to do next

The client can start to sent requests through the firewall to the server that is configured for HTTP tunneling.

Enabling HTTP tunneling

HTTP tunneling enables clients, that reside outside of a firewall, to bundle all of the information, that the client-side Object Request Broker (ORB) needs to send to the server-side ORB, into a normal HTTP request. This request can then be sent to the server on port 80, just like any other HTTP request.

Before you begin

Make sure that the client-side ORB is an IBM ORB. Tunneling does not work if you are using a non-IBM ORB on the client.

Also, if Secure Sockets Layer (SSL) security is required for the tunneling, make sure that the required certificates and key files are configured.

About this task

Sometimes clients residing outside of a firewall need to communicate with modules, such as EJB modules, that reside on a server inside of the firewall. The client-side and server-side ORBs manage this interaction between the client and the server. However, firewalls normally block the ports that a client, uses to talk to the server-side ORB. Therefore if your installation uses a firewall that blocks the ports a client uses to talk to the server-side ORB, you should set up HTTP tunneling.

The IIOPTunnelServlet, which is shipped with the product as class file com.ibm.CORBA.services.IIOPTunnelServlet.class, allows an HTTP client, such as a Java client, that is embedded with RMI-IIOP, to communicate with a server that resides inside of a firewall. This class file, along with the following three class files, are bundled within the WAS_HOME/plugins/com.ibm.ws.runtime_6.1.0.jar file. These additional class files enhance the servlet's capabilities.

com.ibm.CORBA.services.redirector.ConnectionStream.class
com.ibm.CORBA.services.redirector.Redirector.class
com.ibm.CORBA.services.redirector.RedirectorController.class

When tunneling is enabled, the IIOPTunnelServlet servlet on the server receives the HTTP request and unpacks all of the ORB information. The servlet then calls the server-side ORB on the client's behalf. The

server-side ORB treats the request as it would treat any normal ORB request and responds to the servlet. The servlet packs the ORB response into an HTTP response and sends the response back to the client-side ORB, through the firewall. The client-side ORB unpacks the HTTP response and pulls out the response.

Tunneling can operate over HTTPS as well as over HTTP. Therefore, you can use Secure Sockets Layer (SSL) security to secure your tunneling clients if your security procedures require that all communication to your servers is SSL secured.

1. Create an installable IIOPTunnel.ear file that includes the IIOPTunnelServlet servlet.

   Before you can run the IIOPTunnelServlet servlet on the server, you must make it part of an application that you can install on the server. You can use an application assembly tool to create an installable IIOPTunnel.ear file that includes this servlet. For example, if you use the assembly tool that is shipped with the product:

   a. Start the tool.
   b. Open the WEB perspective.
   c. In the Project Explorer view, right click in an empty pane and select **New > Dynamic Web Project**.
   d. In the Create Dynamic Web Project wizard, change the project Name to IIOPTunnel, or another name that is meaningful to you. By default, the **Add Module to an EAR project** option is selected, the EAR project name is set to IIOPTunnelEAR, and the Context Root is set to IIOPTunnel.
   e. Keep these default settings and click **Finish**.
   f. Add the com.ibm.ws.runtime_7.0.0.jar file to the Web Project Build Path.

      Before you can register the new servlet in the Web Deployment Descriptor, you must add the IIOPTunnelServlet servlet, that resides in the WAS_HOME/lib/plugins/com.ibm.ws.runtime_7.0.0.jar file, to your build path.

      1) Right click the IIOPTunnel Web Project, and select **Properties > Java Build Path**.
      2) Select the Libraries tab and press the Add external JARs button.
      3) Add the com.ibm.ws.runtime_7.0.0.jar file, and then click **OK**.

2. Export your EAR file.

   a. Right click on the IIOPTunnelEAR project.
   b. Click **Export > EAR File**, browse to your selected destination directory and specify the EAR file name as IIOPTunnel.ear, or the file name that you specified in Step 1d.
   c. Click **Finish**.

      You get your IIOPTunnel.ear file, which is ready for you to deploy.

3. Install the IIOPTunnel.ear file on your target application server. You can accept all default values during installation.

   Remember to adjust the tunnelAgentURL in the client to reflect the actual location of the IIOPTunnelServlet on your server.

   Note:

   `http(s)://host_name:port/context_root/Servlet_URLmapping`

   The*host_name:port* are the host name and port that is assigned to teh server on which the IIOPTunnelServlet resides. The port can be either an HTTP or an HTTPS port, depending on your security requirements.

   The *context_root* and *Servlet_URLmapping* values must match the values that are defined for the context-root and servlet-URLmapping elements in the servlet web.xml file.

   For example, if the servlet is installed on the default server, and context-root=iioptunnel, and Servlet-URLmapping=tunnel, the following URL must be specified for tunnelAgentURL in the client:

   `http://localhost:9080/IIOPTunnel/IIOPTunnelServlet`

To verify that the servlet is deployed and running successfully, you can open a browser and point to http:// hostname:9080/iioptunnel/tunnel. If the servlet is working, the browser tries to download the servlet as if it were just a normal file. You can then cancel the download.

4. Verify that the servlet is deployed and running successfully

   To verify that the servlet is deployed and running successfully, you can open a browser and point to http:// hostname:9080/IIOPTunnel/IIOPTunnelServlet. If the servlet is working, the browser tries o download the servlet as if it were just a normal file. Simply cancel the download.

   Specify the following parameters if you encounter a problem deploying and running the servlet.

   ```
   -Dcom.ibm.CORBA.TunnelAgentURL=https://localhost:9080/IIOPTunnel/IIOPTunnelServlet?debug=true
   ```

5. Configure the ORB Service for the client-side ORB to enable tunneling

   The client determines whether standard IIOP and HTTP tunneling should be used for communication with the server-side ORB. Therefore you must set the following ORB properties on the client.

   ```
   com.ibm.CORBA.ForceTunnel=ALWAYS
   com.ibm.CORBA.TunnelAgentURL=http://host_name:9080/IIOPTunnel/IIOPTunnelServlet com.ibm.CORBA.FragmentSize=0
   ```

   To enabled tunneling on the client ORB, the com.ibm.CORBA.ForceTunnel property must be set to **ALWAYS**. This setting indicates that this client is always going to tunnel. Other values that can be specified for the com.ibm.CORBA.ForceTunnel property are:

   NEVER, which indicates that you want to disable HTTP tunneling. If a TCP connection fails, a CORBA system exception (COMM_FAILURE) occurs.

   WHENREQUIRED, which indicates that you want to use HTTP tunneling if TCP connections fail.

   The second property specifies the fully qualified URL at which the tunneling servlet is reached. The port 9080 is the WC_defaulthost port for the server. The port number you specify must match the port number that is specified in the configuration file, serverindex.xml, for the server on which the IIOPTunnelServlet servlet resides.

   The third property turns off ORB fragmenting. Normally, the ORB breaks up communications into fragments, to improve performance, but tunneling will not work if the ORB is fragmenting.

   You can also set these properties by adding them as parameters to the JVM command line:

   ```
   -Dcom.ibm.CORBA.ForceTunnel=always
   -Dcom.ibm.CORBA.TunnelAgentURL=http://host_name:9080/iioptunnel/tunnel
   -Dcom.ibm.CORBA.FragmentSize=0
   ```

   Optionally, you can also set the following property if you want to specify client-side security settings:

   ```
   -Dcom.ibm.CORBA.ConfigURL=file:PROFILE_ROOT/properties/sas.client.props
   ```

6. Turn off fragmenting on the server-side ORB. The only property that you must configure for the server-side ORB to enable tunneling is the com.ibm.CORBA.FragmentSize property. This property must be set to `0` to turn off fragmenting.
   a. In the administrative console, click **Servers > Server Types > WebSphere application servers**, and click the server where the tunneling servlet is installed.
   b. Click **ORB Service**, then click **Custom properties**.
   c. Click **New** and then specify **com.ibm.CORBA.FragmentSize** in the Name field and **0** in the Value field.
   d. Click **OK**, and then save the changes.
7. Stop and then restart the application server.

What to do next

The client can start to sent requests through the firewall to the server that is configured for HTTP tunneling.

Chapter 13. Transactions

Using the transaction service

WebSphere Application Server applications can use transactions to coordinate multiple updates to resources as atomic units (as indivisible units of work) such that all or none of the updates are made permanent.

About this task

In WebSphere Application Server, transactions are managed by three main components:

- A transaction manager. The transaction manager supports the enlistment of recoverable XAResources and ensures that each resource of this type is driven to a consistent outcome either at the end of a transaction or after a failure and restart of the application server.
- A container in which the enterprise application runs. The container manages the enlistment of XAResources on behalf of the application when the application performs updates to transactional resource managers (for example, databases). Optionally, the container can control the demarcation of transactions for enterprise beans configured for container-managed transactions.
- An application programming interface, UserTransaction, that is available to bean-managed enterprise beans and servlets. These application components can use the UserTransaction interface to control the demarcation of their own transactions.

  For details about the methods available with the UserTransaction interface, see the Additional Application Programming Interfaces (APIs) or the Java Transaction API (JTA) 1.1 Specification.

  Also, Java Transaction API (JTA) support includes additional application programming interfaces so that application frameworks can manipulate the unit of work (UOW) context of a thread, and components can register with a JTA transaction (for example, a persistence manager can be notified of transaction completion).

Use the following tasks to work with transactions in WebSphere Application Server applications:

- Developing components to use transactions
- “Configuring transaction properties for an application server” on page 2143
- “Managing active and prepared transactions” on page 2155
- “Managing active and prepared transactions using scripting” on page 2156
- “Managing transaction logging for optimum server availability” on page 2159
- “Interoperating transactionally between application servers” on page 2163
- “Using WS-Transaction policy to coordinate transactions or business activities for Web services” on page 688
- Troubleshooting transactions
- Using one-phase and two-phase commit resources in the same transaction
- Using the ActivitySession service

Transaction support in WebSphere Application Server

This topic provides conceptual information about the support for transactions provided by the Transaction Service of WebSphere Application Server.

A transaction is unit of activity, within which multiple updates to resources can be made atomic (as an indivisible unit of work) such that all or none of the updates are made permanent. For example, during the processing of an SQL COMMIT statement, the database manager atomically commits multiple SQL statements to a relational database. In this case, the transaction is contained entirely within the database manager and can be thought of as a *resource manager local transaction (RMLT)*. In some contexts, a transaction is referred to as a *logical unit of work* (LUW). If a transaction involves multiple resource

managers, for example multiple database managers, an external transaction manager is required to coordinate the individual resource managers. A transaction that spans multiple resource managers is referred to as a *global transaction*. WebSphere Application Server is a transaction manager that can coordinate global transactions, can be a participant in a received global transaction, and can also provide an environment in which resource manager local transactions can run.

The way that applications use transactions depends on the type of application component, as follows:
- A session bean can use either container-managed transactions (where the bean delegates management of transactions to the container) or bean-managed transactions (component-managed transactions where the bean manages transactions itself).
- Entity beans use container-managed transactions.
- Web components (servlets) and application client components use component-managed transactions.

WebSphere Application Server is a transaction manager that supports the coordination of resource managers through their XAResource interface, and participates in distributed global transactions with transaction managers that support the CORBA Object Transaction Service (OTS) protocol or Web Service Atomic Transaction (WS-AtomicTransaction) protocol. WebSphere Application Server also participates in transactions imported through Java EE Connector 1.5 resource adapters. You can also configure WebSphere applications to interact with databases, JMS queues, and JCA connectors through their *local transaction* support, when you do not require distributed transaction coordination.

Resource managers that offer transaction support can be categorized into those that support two-phase coordination (by offering an XAResource interface) and those that support only one-phase coordination (for example through a LocalTransaction interface). The WebSphere Application Server transaction support provides coordination, within a transaction, for any number of two-phase capable resource managers. It also enables a single one-phase capable resource manager to be used within a transaction in the absence of any other resource managers, although a WebSphere transaction is not necessary in this case.

Under normal circumstances, you cannot mix one-phase commit capable resources and two-phase commit capable resources in the same global transaction, because one-phase commit resources cannot support the prepare phase of two-phase commit. There are some special circumstances where it is possible to include mixed-capability resources in the same global transaction:
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction and where all the two-phase commit resource-providers that participate in the transaction are used in a read-only fashion. In this case, the two-phase commit resources all vote read-only during the prepare phase of two-phase commit. Because the one-phase commit resource provider is the only provider to actually perform any updates, the one-phase commit resource does not need to be prepared.
- In scenarios where there is only a single one-phase commit resource provider that participates in the transaction with one or more two-phase commit resource providers and where *last participant support* is enabled. Last participant support enables the use of a single one-phase commit capable resource with any number of two-phase commit capable resources in the same global transaction. For more information about last participant support, see Using one-phase and two-phase commit resources in the same transaction.

The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. It is a distributed context that can be used to coordinate multiple one-phase resource managers. The WebSphere EJB container and deployment tooling support ActivitySessions as an extension to the Java EE programming model. Enterprise beans can be deployed with lifecycles that are influenced by ActivitySession context, as an alternative to transaction context. An application can then interact with a resource manager for the period of a client-scoped ActivitySession, rather than only the duration of an EJB method, and have the resource manager's local transaction outcome directed by the ActivitySession. For more information about ActivitySessions, see Using the ActivitySession service.

Resource manager local transaction (RMLT)

A resource manager local transaction (RMLT) is a resource manager's view of a local transaction; that is, it represents a unit of recovery on a single connection that is managed by the resource manager.

Resource managers include:

- Enterprise Information Systems that are accessed through a resource adapter, as described in the Java EE Connector Architecture.
- Relational databases that are accessed through a JDBC datasource.
- JMS queue and topic destinations.

Resource managers offer specific interfaces to enable control of their RMLTs. Resource adapter components of the Java EE connector architecture that include support for local transactions provide a LocalTransaction interface. The LocalTransaction interface enables applications to request that the resource adapter commits or rolls back RMLTs. JDBC datasources provide a Connection interface for the same purpose.

The boundary at which all RMLTs must be complete is defined in WebSphere Application Server by a local transaction containment (LTC).

Global transactions

If an application uses two or more resources, an external transaction manager is needed to coordinate the updates to all the resource managers in a global transaction.

Global transaction support is available to Web and enterprise bean components and, with some limitations, to application client components. Enterprise bean components can be subdivided into two categories: beans that use container-managed transactions (CMT) and beans that use bean-managed transactions (BMT).

BMT enterprise beans, application client components, and Web components can use the Java Transaction API (JTA) UserTransaction interface to define the demarcation of a global transaction. To obtain the UserTransaction interface, use a Java Naming and Directory Interface (JNDI) lookup of java:comp/UserTransaction, or use the getUserTransaction method from the SessionContext object.

The UserTransaction interface is not available to CMT enterprise beans. If CMT enterprise beans attempt to obtain this interface, an exception is thrown, in accordance with the Enterprise JavaBeans (EJB) specification.

Ensure that programs that perform a JNDI lookup of the UserTransaction interface use an InitialContext that resolves to a local implementation of the interface. Also ensure that such programs use a JNDI location that is appropriate for the EJB version.

WebSphere Application Server Version 4 and later releases bind the UserTransaction interface at the JNDI location that is specified in the EJB Version 1.1 specification. This location is java:comp/UserTransaction.

A Web component or enterprise bean (CMT or BMT) can use additional interfaces that provide JTA support. These interfaces provide the transaction identity and a mechanism to receive notification of transaction completion. The interfaces include the TransactionSynchronizationRegistry interface, the ExtendedJTATransaction interface, and the UOWSynchronizationRegistry interface.

Local transaction containment (LTC)

A *local transaction containment (LTC)* is used to define the application server behavior in an unspecified transaction context.

Unspecified transaction context is defined in the Enterprise JavaBeans specification, Version 2.0 and later. For example, see Enterprise JavaBeans Specification.

An LTC is a bounded unit-of-work scope, within which zero, one, or more resource manager local transactions (RMLT) can be accessed. The LTC defines the boundary at which all RMLTs must be complete; any incomplete RMLTs are resolved, according to policy, by the container. By default, an LTC is local to a bean instance; it is not shared across beans, even if those beans are managed by the same container. LTCs are started by the container before dispatching a method on an enterprise application component (such as an enterprise bean or servlet) whenever the dispatch occurs in the absence of a global transaction context. LTCs are completed by the container depending on the application-configured LTC boundary, for example, at the end of the method dispatch. There is no programmatic interface to the LTC support; LTCs are managed exclusively by the container. The application deployer configures LTCs on individual application components (Web application or EJB) by using transaction attributes in the application deployment descriptor.

Note: In this release, a local transaction containment (LTC) is shareable, that is, a single LTC can span multiple application components, including Web application components and enterprise beans that use container-managed transactions, so that those components can share connections without using a global transaction. Sharing a single resource manager between application components improves performance, increases scalability, and reduces lock contention for resources.

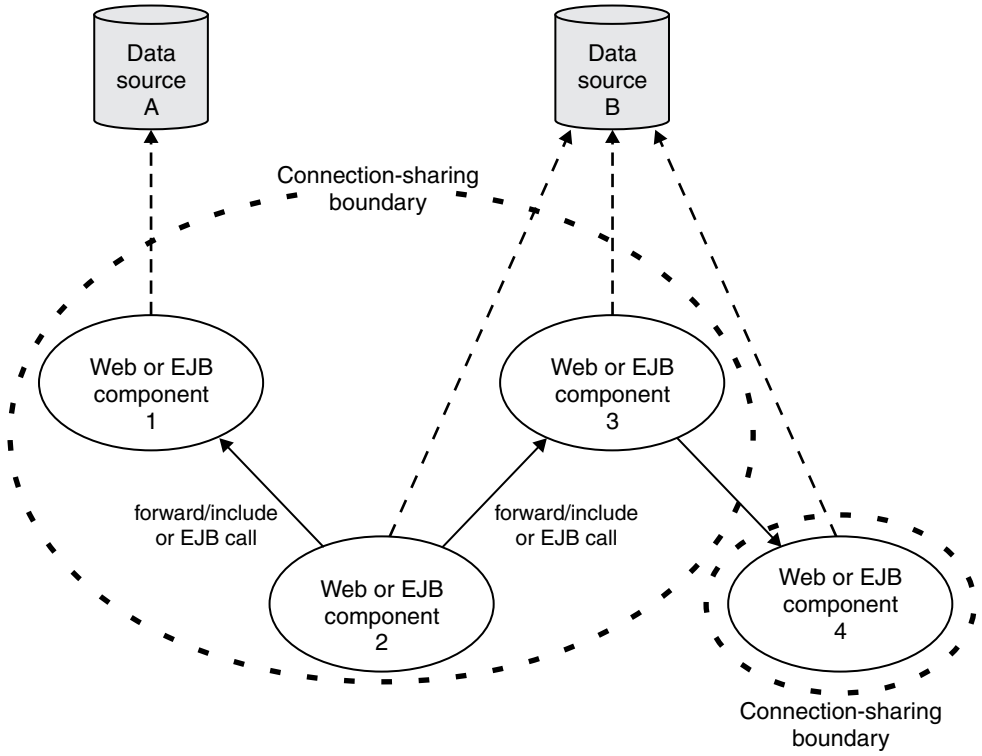
LTCs can be shared across multiple components, including Web application components and enterprise beans that use container-managed transactions. This is useful in situations such as when there is frequent use of Web module include calls, where a thread can have several connections blocked by LTCs in different Web modules. In this situation, the application might encounter code deadlocks under load, when threads start to wait for themselves to free connections. To overcome this issue without using a global transaction, specify that application components can share LTCs by setting the Shareable attribute in the deployment descriptor of each component. You must use a deployment descriptor; you cannot specify this attribute if annotation has been used.

When you set the Shareable attribute, the extended deployment descriptor XML file includes the following line:

```
<local-transaction boundary="BEAN_METHOD" resolver="CONTAINER_AT_BOUNDARY"
unresolved-action="COMMIT" shareable="true"/>
```

To obtain the full benefits of a shared LTC, also ensure that the resource reference for each component defaults to shareable connections.

In the following diagram, components 1, 2 and 3 are deployed with the Shareable attribute and component 4 is not. If components 2 and 3 both obtain connections to data source B, and their resource references for data source B default to shareable connections, they share the connection, but component 4 does not.



Applications that use shareable LTCs cannot explicitly commit or roll back resource manager connections that are used in a shareable LTC (although they can use connections that have an autoCommit capability). This ensures proper encapsulation of connection usage by each component and protects one component from having to make any assumptions about the behavior of other components that share the connection.

If an application starts any non-autocommit work in an LTC for which the Resolver attribute is set to Application and the Shareable attribute is set to true, an exception is thrown at run time. For example, on a JDBC Connection, non-autocommit work is work that the application performs after using the setAutoCommit(false) method to switch off the autocommit option on the connection. Enterprise beans that use bean managed transactions (BMT) cannot be assembled with the Shareable attribute set on the LTC configuration.

A local transaction containment cannot exist concurrently with a global transaction. If application component dispatch occurs in the absence of a global transaction, the container always establishes an LTC for enterprise application components at J2EE 1.3 or later. The only exceptions to this are as follows:

- Application component dispatch occurs without container interposition, for example, for a stateless session bean `create` or a servlet-initiated thread.
- J2EE 1.2 Web components.
- J2EE 1.2 bean-managed transaction (BMT) enterprise beans.

A local transaction containment can be scoped to an ActivitySession context that exists longer than the enterprise bean method in which it is started, as described in ActivitySessions and transaction contexts.

Local and global transaction considerations

Applications use resources, such as Java Database Connectivity (JDBC) data sources or connection factories, that are configured through the Resources view of the administrative console. How these resources participate in a global transaction depends on the underlying transaction support of the resource provider.

For example, most JDBC providers can provide either XA or non-XA versions of a data source. A non-XA data source can support only resource manager local transactions (RMLT), but an XA data source can support two-phase commit coordination, as well as local transactions.

If an application uses two or more resource providers that support only RMLTs, atomicity cannot be assured because of the one-phase nature of these resources. To ensure atomic behavior, the application must use resources that support XA coordination and must access those resources in a global transaction.

If an application uses only one RMLT, atomic behavior can be guaranteed by the resource manager, which can be accessed in a local transaction containment (LTC) context.

An application can also access a single resource manager in a global transaction context, even if that resource manager does not support the XA coordination. An application can do this because the application server performs an “only resource optimization” and interacts with the resource manager in a RMLT. In a global transaction context, any attempt to use more than one resource provider that supports only RMLTs causes the global transaction to be rolled back.

At any moment, an instance of an enterprise bean can have work outstanding in either a global transaction context or a local transaction containment context, but not both. An instance of an enterprise bean can change from running in one type of context to the other (in either direction), if all outstanding work in the original context is complete. Any violation of this principle causes an exception to be thrown when the enterprise bean tries to start the new context.

Client support for transactions

This topic describes the support of application clients for the use of transactions.

Application clients running in an enterprise application client container can explicitly demarcate transaction boundaries, as described in Using component-managed transactions. Application clients cannot perform, directly in the client container, transactional work in the context of any global transaction that they start, because the client container is not a recoverable process.

Application clients can make requests to remote objects, such as enterprise beans, in the context of a client-initiated transaction. Any transactional work performed in a remote, recoverable, server process is coordinated as part of the client-initiated transaction. The transaction coordinator is created on the first server process to which the client-initiated transaction is propagated.

A client can begin a transaction, then, for example, access a JDBC data source directly in the client process. In such cases, any work performed through the JDBC provider is not coordinated as part of the global transaction. Instead, the work runs under a resource manager local transaction. The client container process is non-recoverable and contains no transaction coordinator with which a resource manager can be enlisted.

A client can begin a transaction, then call a remote application component such as an enterprise bean. In such cases, the client-initiated transaction context is implicitly propagated to the remote application server, where a transaction coordinator is created. Any resource managers accessed on the recoverable application server (or any other application server hosting application components invoked by the client) are enlisted in the global transaction.

Client application components need to be aware that locally-accessed resource managers are not coordinated by client-initiated transactions. Client applications acknowledge this through a deployment option that enables access to the UserTransaction interface in the client container. By default, access to the UserTransaction interface in the client container is not enabled. To enable UserTransaction demarcation for an application client component, set the **Allow JTA Demarcation** extension property in the client deployment descriptor. For information about editing the client deployment descriptor, refer to the Rational Application Developer information.

Commit priority for transactional resources

You can specify the order in which transactional resources are processed during two-phase commit processing.

Note: This release introduces resource commit ordering, where you control the order in which transactional resources are processed during two-phase commit processing. Resource commit ordering can increase the occurrence of one-phase commits and therefore improve performance, and reduce problems caused by transaction isolation.

If you control the order in which transactional resources are processed during two-phase commit processing, there are two main benefits:

- One-phase commit optimization occurs more often.
- Potential problems caused by transaction isolation are resolved.

To control the order in which transactional resources are processed during two-phase commit processing, you specify the commit priority of a resource by setting the commit priority attribute on a resource reference. The larger the commit priority, the earlier the resource is processed. For example, if a resource has a commit priority of 10, it is processed before a resource with a commit priority of 1. The commit priority value is of type int and can be between -2147483648 and 2147483647.

If you do not specify a commit priority value, a default value of zero is assigned to the resource and is used when ordering resources at run time. If two or more resources are configured with the same priority, including the default priority, they are processed in an unspecified order with respect to each other.

You can specify the commit priority attribute on a resource reference by using Rational Application Developer tools. For detailed information, see the Rational Application Developer information center. The application component must have a deployment descriptor; you cannot specify this attribute if annotation has been used.

One-phase commit optimization

In a transaction with a two-phase commit, if every resource except the last one enlisted in the transaction votes read-only, indicating that those resources are not interested in the outcome of the transaction, a one-phase commit can occur. This means that the transaction service does not need to store resource and transaction information that it would need to roll back a two-phase commit, and therefore performance is improved.

You can control the order in which transactional resources are processed during two-phase commit, so you can process the resources that are most likely to vote read-only first. Therefore, you increase the chance that a one-phase commit might occur.

Typically, for a given transactional resource, you know the work that is performed at run time, so if you can control the order in which the resources in a transaction are processed, you can increase the likelihood of a one-phase commit optimization occurring.

Transaction isolation

When resources are involved in a global transaction, updates that are made as part of a transaction are not visible outside the transaction until the transaction commits, that is, those resources are isolated. This isolation can cause problems with other application components that act on the updates after they are committed. For example, further processing can fail, or can fail intermittently, because updates are order and time dependent. This problem does not occur with service integration bus messaging work in WebSphere Application Server, but can be a problem for other messaging providers, for example WebSphere MQ.

If you specify the order in which transactional resources are committed, problems caused by isolation are resolved for all transactional systems, not just messaging providers and service integration bus in particular.

The following example describes how problems might occur when you cannot specify the order in which transactional resources are committed. An application updates a row in a database table, then sends a JMS message that triggers additional processing of the row. Both of these actions are performed in the same global transaction, so they are isolated until their respective resources are committed. If the update to the row is committed before the message is sent, the processing that is triggered by the message can access the updated row and process it. If the action to send the message is committed first, this action might trigger the additional processing of the row before the database has committed the update to the row. In this situation, the updated row is still isolated and is not visible, so the additional processing of the row fails.

This problem can be more complicated because it is ordering and timing dependent. If the database is committed first, the problem does not occur. If the action to send the message is committed first, the problem might occur, but it depends whether the database work is committed before the message triggers the further processing of the row. Therefore, the problem can be intermittent, so it is harder to identify its cause.

Considerations with earlier versions of WebSphere Application Server

If you specify the commit priority of a resource, that is, specify any value other than the default value 0, the commit priority is added to the partner log in a recoverable unit section. If you use WebSphere Application Server with versions earlier than WebSphere Application Server Version 7.0, you need to install the appropriate fix pack to the earlier version to ensure that the new section in the log file is recognized. However, commit priority has no effect on the transaction service in versions of WebSphere Application Server that are earlier than Version 7.0.

For general information about different versions of the product, see the topic “Overview of migration, coexistence, and interoperability”.

Transaction compensation and business activity support

A *business activity* is a collection of tasks that are linked together so that they have an agreed outcome. Unlike atomic transactions, activities such as sending an e-mail can be difficult or impossible to roll back atomically, and therefore require a compensation process in the event of an error. The WebSphere Application Server business activity support provides this compensation ability through *business activity scopes*.

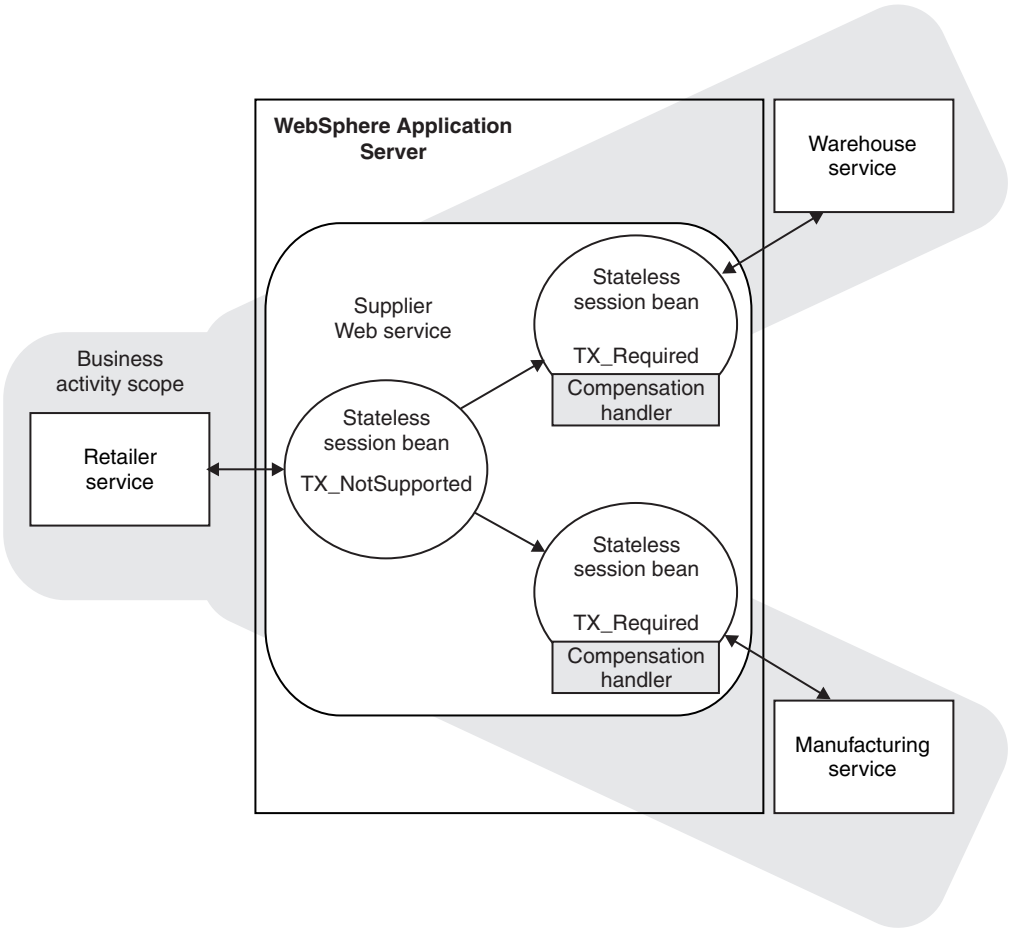
When to use business activity support

Use the business activity support when you have an application that requires compensation. An application requires compensation if its operations cannot be atomically rolled back. Typically, this scenario is because of one of the following reasons:

- The application uses multiple non-extended-architecture (XA) resources.
- The application uses more than one atomic transaction, for example, enterprise beans that have **Requires new** as the setting for the **Transaction** field in the container transaction deployment descriptor.
- The application does not run under a global transaction.

The following diagram shows a simple Web service application that uses the business activity support. The Retailer, Warehouse and Manufacturing services are running in non-WebSphere Application Server environments. The Retailer service calls the Supplier service, running on WebSphere Application Server, which delegates tasks to the Warehouse and Manufacturing services. The implementation of the Supplier service contains a stateless session bean, which calls other stateless session beans that are associated

with the Warehouse and Manufacturing services, and that perform work that can be compensated. These other session beans each have a *compensation handler*; a piece of logic that is associated with an application component at run time, and performs compensation activity such as resending an e-mail.



Application design considerations

Business activity contexts are propagated with application messages, and can therefore be distributed between application components that are not co-located in the same server. Unlike atomic transaction contexts, business activity contexts are propagated on both synchronous (blocking) call-response messages and asynchronous one-way messages. An application component that runs under a business activity scope is responsible for ensuring that any asynchronous work it initiates is complete before the component's own processing is complete. An application that initiates asynchronous work using a fire-and-forget message pattern must not use business activity scopes, because such applications have no means of detecting whether this asynchronous processing has completed.

Only enterprise beans that have container-managed transactions can use the business activity functionality. Enterprise beans that exploit business activity scopes can offer Web service interfaces, but can also offer standard enterprise bean local or remote Java interfaces. Business activity context is propagated in Web service messages using a standard, interoperable Web Services Business Activity (WS-BA) `CoordinationContext` element. WebSphere Application Server can also propagate business activity context on RMI calls to enterprise beans when Web services are not being used, but this form of the context is not interoperable with non-WebSphere Application Server environments. You might want to

use this homogeneous scenario if you require compensation for an application that is internal to your business. If you want to use business activity compensation in a heterogeneous environment, expose your application components as Web services.

Business activity contexts can be propagated across firewalls and outside the WebSphere Application Server domain. The topology that you use to achieve this propagation can affect the high availability and affinity behavior of the business activity transaction.

Application development and deployment considerations

WebSphere Application Server provides a programming model for creating business activity scopes, and for associating compensation handlers with those business activity scopes. WebSphere Application Server also provides an application programming interface to specify compensation data, and check or alter the status of a business activity. To use the business activity support you must set certain application deployment descriptors appropriately, provide a compensation handler class if required, and enable business activity support on any servers that run the application.

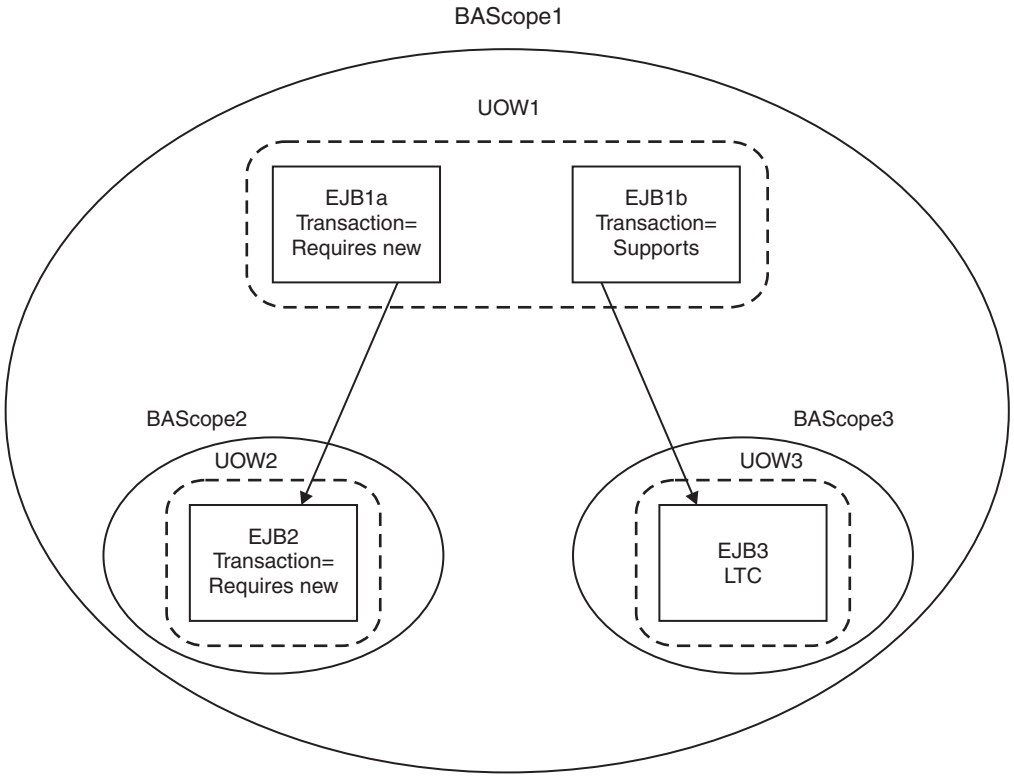
Business activity scopes

The scope of a business activity is that of a core WebSphere Application Server unit of work: a global transaction, an activity session, or local transaction containment (LTC). A business activity scope is not a new unit of work (UOW); it is an attribute of an existing core UOW. Therefore, a one-to-one relationship exists between a business activity scope and a UOW.

In a WS-BA deployment, the UOW must be container-managed:

- The UOW can be a container-managed transaction (CMT) enterprise bean that creates a global transaction.
- The UOW can be a local transaction containment (LTC) where the container is responsible for initiating and ending resource manager local transactions (RMLTs). That is, in the transactional deployment descriptor attributes, the Local Transaction attribute Resolver must be set to ContainerAtBoundary. To use WS-BA, you must not set the Resolver attribute to Application.

Any core UOW can have a business activity scope associated with it. If a component running under a UOW that is associated with a business activity scope calls another component, that request propagates the business activity scope; any work done by the new component is associated with the same business activity scope as the calling component. The called component can create a new UOW, for example if an enterprise bean has a **Transaction** setting of **Requires new**, or runs under the same UOW as the calling component. If a new UOW is started then a new business activity scope is created and associated with the new UOW. The newly created business activity scope is a child of the business activity scope associated with the calling UOW. In the following diagram, EJB1a running under UOW1 calls two components: EJB1b that also runs under UOW1, and EJB2 that creates a new UOW, UOW2. The enterprise bean EJB1b, calls another enterprise bean, EJB3, which creates another new UOW, UOW3. Because each new UOW is created by a calling component whose UOW already has an association with business activity scope BAScope1, the newly created UOWs are associated with new inner business activity scopes, BAScope2 and BAScope3.



Inner business activity scopes must complete before the outer business activity scope completes. Inner business activity scopes, for example BAScope2, have an association with the outer business activity scope, in this case BAScope1. Each business activity scope is directed to close if its associated UOW completes successfully, or to compensate if its associated UOW fails. If BAScope2 completes successfully, any active compensation handlers that are owned by BAScope2 are moved to BAScope1, and are directed in the same way as the completion direction of BAScope1: either compensate or close. If BAScope2 fails, the active compensation handlers are compensated automatically, and nothing is moved to the outer BAScope1. When an inner business activity scope fails, as a result of its associated UOW failing, an application server exception is thrown to the to calling application component, running in the outer UOW.

For example, if the inner UOW fails it might throw a TransactionRolledBackException exception. If the calling application can handle the exception, for example by retrying the called component or by calling another component, then the calling UOW, and its associated business activity scope, can complete successfully even though the inner business activity scope failed. If the application design requires the calling UOW to fail, and for its associated business activity scope to be compensated, then the calling application component must cause its UOW to fail, for example by allowing any system exception from the UOW that failed to be handled by its container.

When the outer business activity scope completes, its success or failure determines the completion direction (close or compensate) of any active compensation handlers that are owned by the outer business activity scope, including those promoted by the successful completion of inner business activity scopes. If the outer business activity scope completes successfully, it drives all active compensation handlers to close. If the outer business activity scope fails, it drives all active compensation handlers to compensate.

This compensation behavior is summarized in the following table.

Table 44. Compensation behavior for a single business activity scope

Inner business activity scope	Outer business activity scope	Compensation behavior
Succeeds	Succeeds	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW succeeds, the outer business activity scope drives all compensation handlers to close.
Fails	Succeeds	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is caught, when the outer UOW succeeds, the outer business activity scope drives all remaining active compensation handlers to close.
Fails	Fails	Any active compensation handlers that are owned by the inner business activity scope are compensated. An exception is thrown to the outer UOW; if this exception is not caught, the outer business activity scope fails. When the outer business activity scope fails, either because of the unhandled exception or for some other reason, all remaining active compensation handlers are compensated.
Succeeds	Fails	Any compensation handlers that are owned by the inner business activity scope wait for the outer UOW to complete. When the outer UOW fails, the outer business activity scope drives all compensation handlers to compensate.

When a UOW with an associated business activity scope completes, the business activity scope always completes in the same direction as the UOW that it is associated with. The only way that you can influence the direction of the business activity scope is to influence the UOW that it is associated with, which you can do by using the setCompensateOnly method of the business activity API.

A compensation handler that is registered within a transactional UOW might initially be inactive, depending on the method invoked from the business activity API. Inactive handlers in this situation become active when the UOW in which that handler is declared completes successfully. A compensation handler that is registered outside a transactional UOW always becomes active immediately. For more information, see “Business activity API” on page 702.

Each business activity scope in the diagram represents a business activity. For example, the outer business activity running under BAScope1 can be a holiday booking scenario, with BAScope2 being a flight booking activity and BAScope3 a hotel booking. If either the flight or hotel bookings fail, the overall holiday booking by default also fails. Alternatively if, for example, the flight booking fails, you might want your application to try booking a flight using another component that represents a different airline. If the overall holiday booking fails, the application can use compensation handlers to cancel any flights or hotels that are already successfully booked.

Use of business activity scopes by application components

Application components do not use business activity scopes by default. You use the WebSphere Application Server assembly tools to specify the use of a business activity scope and to identify any compensation handler class for the component:

Default configuration
: If a business activity context is present on a request received by a component with no business activity scope configuration, the context is stored by the container but never used during the method scope of the target component. A new business activity scope is not created. If the target component invokes another component, the stored business activity context is propagated and can be used by other compensating components.

Run enterprise bean methods under a business activity scope
: Any business activity context present on the incoming request is received by the container and made available to the target component. If a new UOW is created for the target method, for example because the enterprise bean method has a **Transaction** setting of **Requires new**, the received business activity scope becomes an outer business activity scope to a newly created business activity. If the UOW is propagated from the calling component and used by the method,

then the received business activity scope is used by the method. If a business activity scope does not exist on the invocation, a new business activity scope is created and used by the method.

To create a business activity scope when an enterprise bean is invoked, you must configure the enterprise bean to run enterprise bean methods under a business activity scope. You must also configure the deployment descriptors for the method being invoked, to specify the creation of a new UOW upon invocation. For instructions on how to perform these actions, see “Creating an application that uses the Web Services Business Activity support” on page 711.

JTA support

Java Transaction API (JTA) support provides application programming interfaces (APIs) in addition to the UserTransaction interface that is defined in the JTA 1.1 specification.

These interfaces include the TransactionSynchronizationRegistry interface, which is defined in the JTA 1.1 specification, and the following API extensions:

- SynchronizationCallback interface
- ExtendedJTATransaction interface
- UOWSynchronizationRegistry interface
- UOWManager interface

Note: This release features additional Java Transaction API (JTA) support. JTA 1.1 is supported through the introduction of the TransactionSynchronizationRegistry interface. System-level application components, such as persistence managers, resource adapters, enterprise beans, and Web application components, can use the TransactionSynchronizationRegistry interface to register with a JTA transaction. Also, the UOWSynchronizationRegistry interface and the UOWManager interface are introduced so that all types of units of work (UOWs) that the product supports can register with a JTA transaction and can be manipulated.

The APIs provide the following functionality:

- Access to global and local transaction identifiers associated with the thread.

  The global identifier is based on the transaction identifier in the CosTransactions::PropagationContext: object and the local identifier identifies the transaction uniquely in the local Java virtual machine (JVM).
- A transaction synchronization callback that any enterprise application component can use to register an interest in transaction completion.

  Advanced applications can use this callback to flush updates before transaction completion and clear up state after transaction completion. Java EE (and related) specifications position this function typically as the domain of the enterprise application containers.

  Components such as persistence managers, resource adapters, enterprise beans, and Web application components can register with a JTA transaction.

The following information is an overview of the interfaces that the JTA support provides. For more detailed information, see the generated API documentation.

SynchronizationCallback interface

An object implementing this interface is enlisted once through the ExtendedJTATransaction interface, and receives notification of transaction completion.

Although an object implementing this interface can run on a Java platform for enterprise applications server, there is no specific enterprise application component active when this object is called. So, the object has limited direct access to any enterprise application resources. Specifically, the object has no access to the java: namespace or to any container-mediated resource. Such an object can cache a reference to an enterprise application component (for example, a stateless session bean) that it delegates to. The object would then have all the normal access to enterprise application resources. For example, you

could use the object to acquire a Java Database Connectivity (JDBC) connection and flush updates to a database during the beforeCompletion method.

ExtendedJTATransaction interface

This interface is a WebSphere programming model extension to the Java EE JTA support. An object implementing this interface is bound, by enterprise application containers in WebSphere Application Server that support this interface, at java:comp/websphere/ExtendedJTATransaction. Access to this object, when called from an Enterprise JavaBeans (EJB) container, is not restricted to component-managed transactions.

An application uses a Java Naming and Directory Interface (JNDI) lookup of java:comp/websphere/ExtendedJTATransaction to get an ExtendedJTATransaction object, which the application uses as shown in the following example:

```
ExtendedJTATransaction exJTA = (ExtendedJTATransaction)ctx.lookup("
 java:comp/websphere/ExtendedJTATransaction");
SynchronizationCallback sync = new SynchronizationCallback();
exJTA.registerSynchronizationCallback(sync);
```

The ExtendedJTATransaction object supports the registration of one or more application-provided SynchronizationCallback objects. Depending on how the callback is registered, each registered callback is called at one of the following points:

- At the end of every transaction that runs on the application server, whether the transaction is started locally or imported
- At the end of the transaction for which the callback was registered

Note: In this release, the registerSynchronizationCallbackForCurrentTran method is deprecated. Use the registerInterposedSynchronization method of the TransactionSynchronizationRegistry interface instead.

TransactionSynchronizationRegistry interface

This interface is defined in the JTA 1.1 specification. System-level application components, such as persistence managers, resource adapters, enterprise beans, and Web application components, can use this interface to register with a JTA transaction. Then, for example, the component can flush a cache when a transaction completes.

To obtain the TransactionSynchronizationRegistry interface, use a JNDI lookup of java:comp/TransactionSynchronizationRegistry.

Note: Use the registerInterposedSynchronization method to register a synchronization instance, rather than the registerSynchronizationCallbackForCurrentTran method of the ExtendedJTATransaction interface, which is deprecated in this release.

UOWSynchronizationRegistry interface

This interface provides the same functionality as the TransactionSynchronizationRegistry interface, but applies to all types of units of work (UOWs) that WebSphere Application Server supports:

- JTA transactions
- local transaction containments (LTCs)
- ActivitySession contexts

System-level application server components such as persistence managers, resource adapters, enterprise beans, and Web application components can use this interface to register with a JTA transaction. The component can do the following:

- Register synchronization objects with special ordering semantics.

- Associate resource objects with the UOW.
- Get the context of the current UOW.
- Get the current UOW status.
- Mark the current UOW for rollback.

To obtain the UOWSynchronizationRegistry interface, use a JNDI lookup of java:comp/websphere/UOWSynchronizationRegistry. This interface is available only in a server environment.

The following example registers an interposed synchronization with the current UOW:

```
// Retrieve an instance of the UOWSynchronizationRegistry interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWSynchronizationRegistry uowSyncRegistry =
  (UOWSynchronizationRegistry)initialContext.lookup("java:comp/websphere/UOWSynchronizationRegistry");

// Instantiate a class that implements the javax.transaction.Synchronization interface
final Synchronization sync = new SynchronizationImpl();

// Register the Synchronization object with the current UOW.
uowSynchronizationRegistry.registerInterposedSynchronization(sync);
```

UOWManager interface

The UOWManager interface is equivalent to the JTA TransactionManager interface, which defines the methods that allow an application server to manage transaction boundaries. Applications can use the UOWManager interface to manipulate UOW contexts in the product. The UOWManager interface applies to all types of UOWs that WebSphere Application Server supports; that is, JTA transactions, local transaction containments (LTCs), and ActivitySession contexts. Application code can run in a particular type of UOW without needing to use an appropriately configured enterprise bean. Typically, the logic that is performed in the scope of the UOW is encapsulated in an anonymous inner class. System-level application server components such as persistence managers, resource adapters, enterprise beans, and Web application components can use this interface.

WebSphere Application Server does not provide a TransactionManager interface in the API or the system programming interface (SPI). The UOWManager interface provides equivalent functionality, but WebSphere Application Server maintains control and integrity of the UOW contexts.

To obtain the UOWManager interface in a container-managed environment, use a JNDI lookup of java:comp/websphere/UOWManager. To obtain the UOWManager interface outside a container-managed environment, use the UOWManagerFactory class. This interface is available only in a server environment.

You can use the UOWManager interface to migrate a Web application to use Web components rather than enterprise beans, but maintain control over the UOWs. For example, a Web application currently uses the UserTransaction interface to begin a global transaction, makes a call to a method on a session enterprise bean that is configured as not supported to perform some non-transactional work, and then completes the global transaction. You can move the logic that is encapsulated in the session EJB method to the run method of a UOWAction implementation. Then, you replace the code in the Web component that calls the session enterprise bean with a call to the runUnderUOW method of a UOWManager interface to request that this logic is run in a local transaction. In this way, you maintain the same level of control over the UOWs as you had with the original application.

The following example performs some transactional work in the scope of a new global transaction. The transactional work is performed in an anonymous inner-class that implements the run method of the UOWAction interface. Any checked exceptions that the run method creates do not affect the outcome of the transaction.

```
// Retrieve an instance of the UOWManager interface from JNDI.
final InitialContext initialContext = new InitialContext();
final UOWManager uowManager = (UOWManager)initialContext.lookup("java:comp/websphere/UOWManager");

try
```

```
{
 // Invoke the runUnderUOW method, indicating that the logic should be run in a global
 // transaction, and that any existing global transaction should not be joined, that is,
 // the work must be performed in the scope of a new global transaction.
 uowManager.runUnderUOW(UOWSynchronizationRegistry.UOW_TYPE_GLOBAL_TRANSACTION, false, new UOWAction()
 {
   public void run() throws Exception
   {
   // Perform transactional work here.
   }
 });
}

catch (UOWActionException uowae)
{
 // Transactional work resulted in a checked exception being thrown.
}

catch (UOWException uowe)
{
 // The completion of the UOW failed unexpectedly. Use the getCause method of the
 // UOWException to retrieve the cause of the failure.
}
```

Local transaction containment considerations

IBM WebSphere Application Server supports local transaction containment (LTC), which you can configure using local transaction extended deployment descriptors. This topic describes the advantages that this support provides to application programmers, and the relevant settings to use. It also lists points to consider when deciding the best way to configure transaction support for local transactions.

The following sections describe the advantages that LTC support provides, and how to set the local transaction extended deployment descriptors in each situation.

You can develop an enterprise bean or servlet that accesses one or more databases that are independent and require no coordination.

If an enterprise bean does not need to use global transactions, it is often more efficient to deploy the bean with the deployment descriptor for the container transaction type set to NotSupported instead of Required.

With the extended local transaction support of the application server, applications can perform the same business logic in an unspecific transaction context as they can in a global transaction. An enterprise bean, for example, runs in an unspecified transaction context if it is deployed with a container transaction type of NotSupported or Never.

The extended local transaction support provides a container-managed, implicit local transaction boundary, within which the container commits application updates and cleans up their connections. You can design applications with more independence from deployment concerns. This makes using a container transaction type of Supports much simpler, for example, when the business logic might be called either with or without a global transaction context.

An application can follow a get-use-close pattern of connection usage, regardless of whether the application runs in a transaction. The application can depend on the close action behaving in the same way in all situations, that is, the close action does not cause a rollback to occur on the connection if there is no global transaction.

There are many scenarios where ACID coordination of multiple resource managers is not needed. In such scenarios, running business logic in a Transaction policy of NotSupported performs better than in a policy of Required. This benefit is applied through setting the deployment descriptor, in the Local Transactions section, of the Resolver attribute to ContainerAtBoundary. With this setting, application interactions with resource providers, such as databases, are managed within implicit resource manager local transactions (RMLT) that the container both starts and ends. The container commits RMLTs at the containment boundary that is specified by the Boundary attribute in the Local Transactions section; for example, at the end of a method. If the application returns control to the container by an exception, the container rolls back any RMLTs that it has started.

This usage applies to both servlets and enterprise beans.

You can use local transactions in a managed environment that guarantees cleanup.

Applications that want to control RMLTs, by starting and ending them explicitly, can use the default setting of Application for the Resolver extended deployment descriptor in the Local Transactions section. In this situation, the container ensures connection cleanup at the boundary of the local transaction context.

Java platform for enterprise applications specifications that describe application use of local transactions do so in the manner provided by the default settings of Application for the Resolver extended deployment descriptor, and Rollback for the Unresolved action extended deployment descriptor, in the Local Transactions section. When the Unresolved action extended deployment descriptor in the Local Transactions section is set to Commit, the container commits any RMLTs that the application starts but that do not complete when the local transaction containment ends (for example, when the method ends). This usage applies to both servlets and enterprise beans.

You can extend the duration of a local transaction beyond the duration of an EJB component method.

The Enterprise JavaBeans (EJB) specifications restrict the use of RMLTs to single EJB methods. This restriction is because the specifications have no scoping device, beyond a container-imposed method boundary, to which an RMLT can be extended. You can use the Boundary extended deployment setting in the Local Transactions section to give the following advantages:

- Significantly extend the use cases of RMLTs.
- Make conversational interactions with one-phase resource managers possible through ActivitySession support.

You can use an ActivitySession to provide a distributed context with a boundary that is longer than a single method. You can extend the use of RMLTs over the longer ActivitySession boundary, which a client can control. The ActivitySession boundary reduces the need to use distributed transactions where ACID operations on multiple resources are not needed. This benefit is applied through the Boundary extended deployment setting, in the Local transactions section, of ActivitySession. Such extended RMLTs can remain under the control of the application, or be managed by the container, depending on the setting of the Resolver deployment descriptor in the Local Transactions section.

You can coordinate multiple one-phase resource managers.

For resource managers that do not support XA transaction coordination, a client can use ActivitySession-bounded local transaction contexts. Such contexts give a client the same ability to control the completion direction of the resource updates by the resource managers as the client has for transactional resource managers. A client can start an ActivitySession and call its entity beans in that context. Those beans can perform their RMLTs within the scope of that ActivitySession and return without completing the RMLTs. The client can later complete the ActivitySession in a commit or rollback direction and cause the container to drive the ActivitySession-bounded RMLTs in that coordinated direction.

You can use shareable LTCs to reduce the number of connections you require.

Application components can share LTCs. If components obtain connections to the same resource manager, they can share that connection if they run under the same global transaction or shareable LTC. To configure two components to run under the same shareable LTC, set the Shareable attribute of the Local Transactions section in the deployment descriptor of each component. Make sure that the resource reference in the deployment descriptor for each component uses the default value of Shareable for the res-sharing-scope element, if this element is specified. A shareable LTC can reduce the numbers of RMLTs an application uses. For example, an application that makes frequent use of Web module include calls can share resource manager connections between those Web modules, exploiting either shareable LTCs, or a global transaction, reducing lock contention for resources.

Examples of local transaction support configurations

The following list gives scenarios that use local transactions, and points to consider when deciding the best way to configure the transaction support for an application.

- You want to start and end global transactions explicitly in the application (bean-managed transaction session beans and servlets only).

  For a session bean, set the Transaction type to Bean (to use bean-managed transactions) in the deployment descriptor of the component. You do not need to do this for servlets.
- You want to access only one XA or non-XA resource in a method.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to Supports.
- You want to access several XA resources atomically across one or more bean methods.

  In the deployment descriptor of the component, in the Container Transactions section, set the container transaction type to Required, RequiresNew, or Mandatory.
- You want to access several non-XA resources in a method without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary. In the Container Transactions section, set the container transaction type to NotSupported.
- You want to access several non-XA resources in a method and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Unresolved action attribute to Rollback. In the Container Transactions section, set the container transaction type to NotSupported.
- You want to access one or more non-XA resources across multiple EJB method calls without needing to manage your own local transactions.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to ContainerAtBoundary and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.
- You want to access several non-XA resources across multiple EJB method calls and want to manage them independently.

  In the deployment descriptor of the component, in the Local Transactions section, set the Resolver attribute to Application and set the Boundary attribute to ActivitySession. In the Bean Cache section, set the Activate at attribute to ActivitySession. In the Container Transactions section, set the container transaction type to NotSupported and set the ActivitySession kind attribute to Required, RequiresNew, or Mandatory.

Transaction service exceptions

The exceptions that the WebSphere Application Server transaction service can throw are listed with a summary of each exception.

The exceptions are grouped as follows:
- Standard exceptions
- Heuristic exceptions

If the EJB container catches a system exception from the business method of an enterprise bean, and the method is running within a container-managed transaction, the container rolls back the transaction before passing the exception on to the client. For more information about how the container handles the exceptions thrown by the business methods for beans with container-managed transaction demarcation, see the section *Exception handling* in the Enterprise JavaBeans 2.0 specification. That section specifies the container's action as a function of the condition under which the business method executes and the exception thrown by the business method. It also illustrates the exception that the client receives and how the client can recover from the exception.

Standard exceptions

The standard exceptions such as TransactionRequiredException, TransactionRolledbackException, and InvalidTransactionException are defined in the Java Transaction API (JTA) 1.1 specification.

InvalidTransactionException
This exception indicates that the request carried an invalid transaction context.

TransactionRequiredException exception
This exception indicates that a request carried a null transaction context, but the target object requires an active transaction.

TransactionRolledbackException exception
This exception indicates that the transaction associated with processing of the request has been rolled back, or marked for roll back. Thus the requested operation either could not be performed or was not performed because further computation on behalf of the transaction would be fruitless.

Heuristic exceptions

A heuristic decision is a unilateral decision made by one or more participants in a transaction to commit or rollback updates without first obtaining the consensus outcome determined by the Transaction Service. Heuristic decisions are an issue only after the participant has been prepared and the second phase of commit processing is underway. Heuristic decisions are normally made only in unusual circumstances, such as repeated failures by the transaction manager to communicate with a resource manage during two-phase commit. If a heuristic decision is taken, there is a risk that the decision differs from the consensus outcome, resulting in a loss of data integrity.

The following list provides a summary of the heuristic exceptions. For more detail, see the Java Transaction API (JTA) 1.1 specification.

HeuristicRollback exception
This exception is raised on the commit operation to report that a heuristic decision was made and that all relevant updates have been rolled back.

HeuristicMixed exception
This exception is raised on the commit operation to report that a heuristic decision was made and that some relevant updates have been committed and others have been rolled back.

Configuring transaction properties for an application server

You can view or change settings for the transaction service. For example, you can change the location or default file size of the transaction log files, change transaction timeout properties, or change heuristic-related properties.

About this task

The transaction service is a server runtime component that can coordinate updates to multiple resource managers to ensure atomic updates of data. Transactions are started and ended by applications or the container in which the applications are deployed.

You might perform this task when you want to move the transaction logs to a different storage device, or when you need to change the transaction service settings. You must restart the application server to make configuration changes take effect.

Note: In this release, both the WS-Transaction 1.1 and the WS-Transaction 1.0 specifications are supported. You can set the default WS-Transaction specification level for a server to use for outbound requests that include a Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) coordination context.

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name***. The properties of the application server, *server_name*, are displayed in the content pane.

2. Click **[Container Settings] Container Services → Transaction Service**. The Transaction Service settings page is displayed.
3. Ensure that the Configuration tab is displayed.
4. Optional: To change the directory in which transaction logs are written, type the full path name of the directory in the **Transaction log directory** field. You can check the current runtime value of **Transaction log directory** by clicking the **Runtime** tab.

   If do not set this field, the application server assumes a default location in the appropriate profile directory.

   Note: If you change the transaction log directory, apply the change and restart the application server as soon as possible, to minimize the risk of problems occurring before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server starts again with the new log directory and cannot automatically resolve in-flight transactions that were recorded in the previous log directory.

   You can also specify a size for the transaction logs, as described in the following step.
5. Optional: To change the default file size of transaction log files, modify the **Transaction log directory** field to include a file size setting. Use the following format, where *directory_name* is the name of the transaction log directory and *file_size* is the new default size specified in KB (*n*K) or MB (*n*M). If you do not specify a value for the file size, the default value of 1M is used.

directory_name;*file_size*

   In a non-production environment, you can turn transaction logging off by entering `;0` in the **Transaction log directory** field (do not enter a directory name). Do not turn transaction logging off in a production environment because this prevents recovery after a system failure, and therefore data integrity cannot be guaranteed.

   For more information about transaction log sizes, see "Managing transaction logging for optimum server availability" on page 2159.
6. Optional: Review or change the value of transaction timeout properties:

   Total transaction lifetime timeout
   The number of seconds to allow for a transaction that is started on this server, before the transaction service initiates timeout completion. If a transaction does not begin completion processing before this timeout occurs, it is rolled back. A value of `0` (zero) indicates that this timeout does not apply, and therefore the maximum transaction timeout is used instead. Application components can override the total transaction lifetime timeout for their transactions by setting their own timeout value.

   Maximum transaction timeout
   The number of seconds a transaction that is propagated into this application server can remain inactive before it is ended by the transaction service. This value also applies to transactions that are started in this server, if their associated applications do not set a transaction timeout and the total transaction lifetime timeout is set to `0`.

   This value must be equal to, or greater than, the total transaction lifetime timeout. A value of `0` (zero) indicates that this timeout does not apply. In this situation, transactions that are affected by this timeout never time out.

   Client inactivity timeout
   The number of seconds after which a client is considered inactive and the transaction service ends any transactions associated with that client. A value of `0` (zero) indicates that there is no timeout limit.
7. Optional: Review or change heuristic-related properties:

   Heuristic retry limit
   The number of times that the application server retries a completion signal, such as commit or rollback. Retries occur after a transient exception from a resource manager or remote

partner, or if the configured asynchronous response timeout expires before all Web Services Atomic Transaction (WS-AT) partners have responded.

Heuristic retry wait
The number of seconds that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

Enable logging for heuristic reporting
Select this option to enable the application server to log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Heuristic completion direction
Select the direction used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator.

Accept heuristic hazard
Select this option to specify that all applications on this server accept the possibility of a heuristic hazard occurring in a two-phase transaction that contains a one-phase resource. This setting configures last participant support (LPS) for the server. If you do not select this option, you must configure applications individually to accept the heuristic hazard.

8. Optional: To change the default WS-Transaction specification level to use for outbound requests that include a Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) coordination context, select the specification level from the **Default WS-Transaction specification level** list.
9. Review or change other configuration properties, to suit your requirements. For more information about the properties of the transaction service, see “Transaction service settings.”
10. Click **OK**, then save your changes to the master configuration.
11. Stop, then restart, the application server.

What to do next

If you change the transaction log directory configuration property to an incorrect directory name, the application server restarts, but cannot open the transaction logs. Change the configuration property to a valid directory name, then restart the application server.

If you are running the application server as non-root, modify the permissions on the new transaction log location. To use peer recovery of transactions on a shared device with non-root users, make sure that your non-root users and groups have matching identification numbers across machines.

Transaction service settings

Use this page to specify settings for the transaction service. The transaction service is a server runtime component that can coordinate updates to multiple resource managers to ensure atomic updates of data. Transactions are started and ended by applications or the container in which the applications are deployed.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Container Settings] Container Services** → **Transaction Service**.

Transaction log directory

Specifies the name of a directory for this server where the transaction service stores log files for recovery. Optionally, you can specify the size of transaction log files.

Set this property to change the log file directory for an application server only if the applications use distributed resources or XA transactions; for example, multiple databases and resources are accessed in a single transaction.

If you do not specify this directory during server configuration, the transaction service uses a default directory that is based on the installation directory: *app_server_root*/ tranlog/*cell_ name*/*node_ name*/*server_ name*.

When an application that runs on the application server accesses more than one resource, the application server stores transaction information in the product directory so that it can coordinate and manage the distributed transaction correctly. When there is a higher transaction load, storing persistent information in this way can slow the performance of the application server because it depends on the operating system and the underlying storage systems. To achieve better performance, designate a new directory for the log files on a separate, physically larger, storage system.

If your application server demonstrates one or more of the following symptoms, change the transaction log directory:
- CPU use remains low despite an increase in transactions
- Transactions fail with several timeouts
- Transaction rollbacks occur with the exception “Unable to enlist transaction”
- The application server stops in the middle of a run and must be restarted
- The disk that the application server is running on shows higher use

There are the following recommendations for a storage system for the log files:
- Store log files on a redundant array of independent disks (RAID)

  In RAID configurations, the task of writing data to the physical media is shared across the multiple drives. This technique yields more concurrent access to storage for persisting transaction information, and faster access to that data from the logs. Depending on the design of the application and storage subsystem, performance gains can range from 10% to 100%, or more in some cases.
- Do not store log files with the operation system I/O mode set to concurrent I/O (CIO)

  When you designate a transaction log directory, ensure that the file system uses only synchronous write-through and write serialization operations. Some operating systems, such as AIX JFS2, support an optional concurrent I/O (CIO) mode, where the file system does not enforce serialization of write operations. On these systems, do not use CIO mode for application server transaction recovery log files.

To specify the size of transaction log files, include a file size setting. Use the following format, where *directory_name* is the name of the transaction log directory and *file_size* is the new disk space allocation for the transaction log files, specified in KB (*n*K) or MB (*n*M). The minimum transaction log file size that you can specify is 64K. If you specify a value that is less than 64K, or you do not specify a value for the file size, the default value of 1M is used.

directory_name;*file_size*

For more information about transaction log sizes, see “Managing transaction logging for optimum server availability” on page 2159.

Data type	String
Default	Directory name: *app_server_root*/tranlog/*cell_name*/*node_name*/*server_name* File size: 1MB

Recommended	Create a file system with at least three to four disk drives raided together in a RAID-0 configuration. Then, create the transaction log on this file system with the default size. When the server is running under load, check the disk input and output. If disk input and output time is more then 5%, consider adding more physical disks to lower the value.

If you migrate a WebSphere Application Server Version 5 node to Version 6, the stored location of this configuration property is moved from the server level to the node (server index) level. If you specified a non-default log directory for a Version 5 application server, you are prompted to save the transaction service settings again, to confirm that you want the log directory saved to the node level.

Total transaction lifetime timeout

The default maximum time, in seconds, allowed for a transaction that is started on this server before the transaction service initiates timeout completion. Any transaction that does not begin completion processing before this timeout occurs is rolled back.

This timeout is used only if the application component does not set its own transaction timeout.

The upper limit of this timeout is constrained by the maximum transaction timeout. For example, if you set a value of `500` for the total transaction lifetime timeout, and a value of `300` for the maximum transaction timeout, transactions will time out after 300 seconds.

If you set this timeout to `0`, the timeout does not apply and the value of the maximum transaction timeout is used instead.

Data type	Integer
Units	Seconds
Default	120
Range	0 to 2 147 483 647

Asynchronous response timeout

Specifies the amount of time, in seconds, that the server waits for an inbound Web Services Atomic Transaction (WS-AT) protocol response before resending the previous WS-AT protocol message.

Data type	Integer
Units	Seconds
Default	30
Range	0 to 2 147 483 647

Client inactivity timeout

Specifies the maximum duration, in seconds, between transactional requests from a remote client. Any period of client inactivity that exceeds this timeout results in the transaction being rolled back in this application server.

If you set this value to `0`, there is no timeout limit.

Data type	Integer
Units	Seconds
Default	60
Range	0 to 2 147 483 647

Maximum transaction timeout

Specifies, in seconds, the upper limit of the transaction timeout for transactions that run in this server. This value should be greater than or equal to the value specified for the total transaction timeout.

This timeout constrains the upper limit of all other transaction timeouts. The following table shows how the different timeout settings apply to transactions running in the server.

Table 45. Transaction timeout settings.

Timeout setting	Transactions affected
Maximum transaction timeout	All transactions running in this server that are not affected by the total transaction lifetime timeout or an application component timeout. These transactions include transactions imported from outside this server, such as those imported from a client.
Total transaction lifetime timeout	All transactions that originated in this server that are not affected by an application component timeout, in other words, the associated application component does not set its own timeout.
Application component timeout	Transactions that are specific to an application component. If the component is a container-managed bean, set this timeout in the deployment descriptor for the component. If the component is a bean-managed bean, set this timeout programmatically using the UserTransaction.setTransactionTimeout method.

If you set a timeout to `0`, that timeout does not apply, and is effectively disabled. If you set all timeouts to `0`, transactions never time out.

For example, consider the following timeout values:

Table 46. Example timeout values

Timeout setting	Value
Maximum transaction timeout	360
Total transaction lifetime timeout	240
Application component timeout	60

In this example, transactions that are specific to the application component time out after 60 seconds. Other local transactions time out after 240 seconds, and any transactions that are imported from outside this server time out after 360 seconds. If you then change the application component timeout to `500`, application component transactions time out after 360 seconds, the value of the maximum transaction timeout. If you set the maximum transaction timeout to `0`, application component transactions time out after 500 seconds. If you remove the application component timeout, application component transactions time out after 240 seconds.

To determine the occurrence of a timeout quickly, and to prevent further resource locking, the application server prevents further transactional work on the transactional path where the timeout condition has taken place. This applies equally to attempting to perform work under the current transaction context and to attempting to perform work under a different transactional context.

Data type Integer
Units Seconds
Default 300
Range 0 to 2 147 483 647

Heuristic retry limit

Specifies the number of times that the application server retries a completion signal, such as commit or rollback. Retries occur after a transient exception from a resource manager or remote partner, or if the configured asynchronous response timeout expires before all Web Services Atomic Transaction (WS-AT) partners have responded.

If the application server abandons the retries, the resource manager or remote partner is responsible for ensuring that the resource or partner's branch of the transaction is completed appropriately. The application server raises (on behalf of the resource or partner) an exception that indicates a heuristic hazard. If a commit request was made, the transaction originator receives an exception on the commit

operation; if the transaction is container-initiated, the container returns a remote exception or Enterprise JavaBeans (EJB) exception to the EJB client.

Data type Integer
Default 0
Range 0 to 2 147 483 647

A value of 0 (the default) means retry indefinitely.

Heuristic retry wait

Specifies the number of seconds that the application server waits before retrying a completion signal, such as commit or rollback, after a transient exception from a resource manager or remote partner.

Data type Integer
Default 0
Range 0 to 2 147 483 647

A value of 0 means that the application server determines the retry wait; the server doubles the retry wait after every 10 failed retries.

Enable logging for heuristic reporting

Specifies whether the application server logs about-to-commit-one-phase-resource events from transactions that involve both a one-phase commit resource and two-phase commit resources.

This property enables logging for heuristic reporting. If applications are configured to allow one-phase commit resources to participate in two-phase commit transactions, reporting of heuristic outcomes that occur at application server failure requires extra information to be written to the transaction log. If enabled, one additional log write is performed for any transaction that involves both one-phase and two-phase commit resources. No additional records are written for transactions that do not involve a one-phase commit resource.

Data type Check box
Default Cleared
Range

Cleared
The application server does not log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Selected
The application server does log ″about to commit one-phase resource″ events from transactions that involve a one-phase commit resource and two-phase commit resources.

Heuristic completion direction

Specifies the direction that is used to complete a transaction that has a heuristic outcome; either the application server commits or rolls back the transaction, or depends on manual completion by the administrator.

Data type Drop-down list
Default ROLLBACK

Range

COMMIT
The application server heuristically commits the transaction.

ROLLBACK
The application server heuristically rolls back the transaction.

MANUAL
The application server depends on an administrator to manually complete or roll back transactions with heuristic outcomes.

Accept heuristic hazard

Specifies whether all applications on this server accept the possibility of a heuristic hazard occurring in a two-phase transaction that contains a one-phase resource. This setting configures last participant support (LPS) for the server. Last participant support is an extension to the transaction service that enables a single one-phase resource to participate in a two-phase transaction with one or more two-phase resources.

If the Accept heuristic hazard option is not selected, you must configure applications individually to accept the heuristic hazard. You can configure applications either when they are assembled, or following deployment by using the Last participant support extension pane.

Data type Check box
Default Cleared
Range

Selected
All applications deployed on the server accept the increased risk of an heuristic outcome.

Cleared
Applications must be individually configured to accept the increased risk of an heuristic outcome.

Enable file locking

Specifies whether the use of file locks is enabled when opening the transaction service recovery log.

If you enable this setting, a file lock will be obtained before accessing the transaction service recovery log files. File locking is used to ensure that, in a highly available WebSphere Application Server deployment, only one application server can access a particular transaction service recovery log at any one time. This setting has no effect in a standard deployment where you have not configured high availability support.

Note: This setting requires a compatible network file system, such as Network File System (NFS) version 4, to operate correctly.

Data type Check box
Default Selected

Enable transaction coordination authorization

Specifies whether the secure exchange of transaction service protocol messages is enabled.

This setting has no effect unless you enable WebSphere Application Server security on the server.

Data type Check box
Default Selected

Default WS-Transaction specification level

Specifies the default WS-Transaction specification level to use for outbound requests that include a Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) coordination context.

You can choose from WS-Transaction 1.1 or WS-Transaction 1.0. For details of these specifications, see "Web Services Atomic Transaction support in the application server" on page 689 and "Web Services Business Activity support in the application server" on page 693.

The default WS-Transaction specification level is used if the specification level that the server requires cannot be determined from the provider policy (the WS-Transaction WS-Policy assertion). For example, the policy assertion is not available, either from the WSDL of the target Web service or from the WS-Transaction policy type of the client, or the policy assertion is available but both specification levels are applicable.

Data type	Drop-down list
Default	1.0

External WS-Transaction HTTP(S) URL prefix

Select or specify the external WS-Transaction HTTP(S) URL prefix.

Select or specify one of these fields if you are using an intermediary node, such as an HTTP server or Proxy Server for WebSphere, to send requests that comply with the Web Services Atomic Transaction (WS-AT) or Web Services Business Activity (WS-BA) protocols.

If WebSphere Application Server security is enabled and transaction coordination authorization is enabled, the HTTPS prefix is used. Otherwise, the HTTP prefix is used.

If the intermediary node is not a Proxy Server, the prefix must be unique for each server.

Select prefix

Select this option to select the external endpoint URL information to use for WS-AT and WS-BA service endpoints from the list.

Data type	Drop-down list
Default	None

Specify custom prefix

Select this option to specify the external endpoint URL information to use for WS-AT and WS-BA service endpoints in the field.

Use one of the following formats for the prefix, where *host_name* and *port* represent the intermediary node that is an HTTP or HTTPS proxy for the server.

`http://host_name:port`

`https://host_name:port`

Data type	String
Default	None

Manual transactions

Specifies the number of transactions that await manual completion by an administrator.

If there are transactions awaiting manual completion, you can click the **Review** link to display a list of those transactions on the Transactions needing manual completion panel.

Data type	Integer
Default	0

Retry transactions
Specifies the number of transactions with some resources being retried.

If there are transactions with resources being retried, you can click the **Review** link to display a list of those transactions on the Transactions retrying resources panel.

Data type	Integer
Default	0

Heuristic transactions
Specifies the number of transactions that have completed heuristically.

If there are transactions that have completed heuristically, you can click the **Review** link to display a list of those transactions on the Transactions with heuristic outcome panel.

Data type	Integer
Default	0

Imported prepared transactions
Specifies the number of transactions that are imported and prepared but not yet committed.

If there are transactions that have been imported and prepared but not yet committed, you can click the **Review** link to display a list of those transactions on the Transactions imported and prepared panel.

Data type	Integer
Default	0

Transactions needing manual completion
Use this page to review transactions that need manual completion.

It is unusual for transactions to require manual completion. A transaction needs manual completion in the following circumstances:

1. An application was exploiting the last participant support to coordinate a single one-phase capable resource and one or more two-phase capable resources.
2. A failure occurred during the commit of the one-phase capable resource.
3. The transaction service heuristic completion direction is set to **Manual**, in the transaction service settings.

An administrator reviewing transactions in this state can review the actual outcome of any one-phase resources, using facilities provided by the specific resource manager, then use this page to complete the transaction accordingly.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Manual transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Commit
Heuristically commit the selected transactions.

Rollback
Heuristically roll back the selected transactions.

Transactions retrying resources

Use this page to review transactions with resources being retried.

If the transaction manager has prepared resources, but has lost contact with the resource managers before committing them or aborting them, then the transaction manager retries the commit or rollback requests to the affected resource managers. The number of times and frequency that the transaction manager retries such commit or rollback requests is configured on the **Heuristic retry limit** and **Heuristic retry wait** properties of the transaction service settings.

An administrator can use this page to make the transaction service abandon the retries of one or more transactions. If a resource manager cannot be contacted, WebSphere Application Server relegates that resource manager to an in-doubt (prepared) state. The administrator then needs to use mechanisms specific to the resource manager to resolve the in-doubt status.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers >** [Content pane] *server_name* **> [Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Retry transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Status
The status of the transaction, shown as an integer value. The values correspond to the following status:

0 - active
1 - marked for rollback
2 - prepared
3 - committed
4 - rolled back
5 - unknown
6 - none
7 - preparing
8 - committing
9 - rolling back

Global ID
The global identifier of the transaction.

Buttons

Finish Abandon retrying resources for the selected transactions.

Transactions with heuristic outcome

Use this page to review transactions that completed with a heuristic outcome.

The page is provided for information purposes only. After you have reviewed the information in this page, then the only action required is to remove the transactions from the list. If you do not remove a transaction from the list, it is kept in the list for three days or until the server is shut down, whichever occurs first.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** [Content pane] *server_name* [Container Settings] **Container Services** → **Transaction Service** → **Runtime** → **Heuristic transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Heuristic outcome
The outcome of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Clear Remove the selected transactions from the list.

Transactions imported and prepared

Use this page to review transactions that have been imported and prepared but not yet committed.

Under normal circumstances no administrative action is required for any of the transactions listed on this page. This page lists those transactions that are in a prepared state, but are being directed by an external transaction manager (for example, another WebSphere application server) from a transaction context that has been propagated.

Under aberrant circumstances, however, an administrator can configure WebSphere Application Server to resolve the transactions listed on this page independent of the external transaction manager. (This step might be necessary if, for example, the external transaction manager has become unavailable for an unacceptable period of time.)

Note: If the completion direction (commit or rollback) chosen administratively differs from the eventual direction of the external transaction manager, then the overall outcome of the transaction is not atomic and data corruption can result.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → **Imported prepared transactions - Review**.

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, click the check boxes next to the transactions that you want to act on, then use the buttons provided.

Local ID
The local identifier of the transaction.

Global ID
The global identifier of the transaction.

Buttons

Commit
Heuristically commit the selected transactions.

Rollback
Heuristically roll back the selected transactions.

Transaction resources
Use this page to review resources used by a transaction.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** [Content pane] *server_name* → **[Container Settings] Container Services** → **Transaction Service** → **Runtime** → *transaction_type local_ID*.

Where:

- *transaction_type* is one of:
  - **Manual transactions - Review**
  - **Retry transactions - Review**
  - **Heuristic transactions - Review**
  - **Imported prepared transactions - Review**
- *local_ID* is the local ID of the transaction (as an active link in the list of transactions).

The details displayed depend on the resource provider.

Managing active and prepared transactions
Use this task to manage active and prepared transactions that might need administrator action.

About this task

Under normal circumstances, transactions should run and complete (commit or roll back) automatically, without the need for intervention. However, in some circumstances, you might need to resolve a transaction manually. For example, you might want to roll back a transaction that has become stuck polling a resource manager that you know will not become available again within the desired timeframe.

Note: If you choose to complete a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so will not be eligible for recovery during server start up. If you complete a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

You can use the administrative console to display a snapshot of all the transactions in an application server that are in the following states:

Manual transactions
Transactions awaiting administrative completion. For each transaction, the local id or global id is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to commit or roll back transactions in this state.

Retry transactions
Transactions with some resources being retried. For each transaction, the local id or global id is displayed, and whether the transaction is committing or rolling back. You can choose to display

information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to finish (abandon retrying) transactions in this state.

Heuristic transactions
Transactions that have completed heuristically. For each transaction, the local id or global id and the heuristic outcome is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to clear the transaction from the list.

Imported prepared transactions
Transactions that have been imported and prepared but not yet committed. For each transaction, the local id or global id is displayed. You can choose to display information on each resource (specifically, which resource manager it is associated with) associated with the transaction. You can also choose to commit or roll back transactions in this state.

To manage the active and prepared transactions for an application server, use the administrative console to complete the following steps:

1. Display the Transaction Service runtime page for application server:
   a. In the navigation pane, click **Servers** → **Server Types** → **WebSphere application servers**.
   b. In the content pane, click the name of the application server.
   c. In the content pane, click the **Runtime** tab.
   d. Under Additional Properties, click **Transaction Service**.

   The page displays values for the runtime properties of the transaction service, including the number of transactions in the active and prepared states.
2. To display a snapshot of the transactions in a specific state, click **Review** in the field label.
3. Optional: To display information about the resources associated with a transaction, click the name of the transaction.
4. Optional: To act on a transaction, select the check box provided on the entry for the transaction, then click one of the buttons provided. Alternatively, to act on all transactions, select the check box in the header of the transactions table, then click a button.

Managing active and prepared transactions using scripting

You can use scripting to manage active and prepared transactions that might need administrator action.

Before you begin

Before you start this task, the wsadmin tool must be running. See Starting the wsadmin scripting client for more information.

About this task

In normal circumstances, transactions should run and complete (commit or roll back) automatically, without the need for intervention. However, in some circumstances, you might need to resolve a transaction manually. For example, you might want to roll back a transaction that is stuck polling a resource manager that you know will not become available again in the desired time frame.

Note: If you choose to complete a transaction on an application server, it is recorded as having completed in the transaction service logs for that server, so it is not eligible for recovery during server start up. If you complete a transaction, you are responsible for cleaning up any in-doubt transactions on the resource managers affected.

The TransactionService Managed Bean (MBean), see API documentation - Application programming interfaces (package: Public MBean Interfaces, class: TransactionService), is used to list transactions in various states by invoking one of the following methods:

- listOfTransactions: Lists all non-completed transactions. Under no circumstances should you attempt to alter the state of active transactions (using commit or rollback for example)
- listManualTransactions: Lists transactions awaiting administrative completion. You can choose to commit or rollback transactions in this state
- listRetryTransactions: Lists transactions with some resources being retried. You can choose to finish (abandon retrying) transactions in this state
- listHeuristicTransactions: Lists transactions that have completed heuristically. You can choose to clear the transaction from the list
- listImportedPreparedTransactions: Lists transactions that have been imported and prepared but not yet committed. You can choose to commit or rollback transactions in this state

Each entry in the returned list contains the following attributes:

- **Local Transaction Identifier**
- **Status**, which can be interpreted by calling getPrintableStatus on the Transaction MBean
- **Global Transaction Identifier**
- **Heuristic Outcome**, which can take one of the following values:
  - 8 (HEURISTIC_COMMIT)
  - 9 (HEURISTIC_ROLLBACK)
  - 10 (HEURISTIC_MIXED)
  - 11 (HEURISTIC_HAZARD)

The TransactionService MBean, see API documentation - Application programming interfaces (package: Public MBean Interfaces, class: TransactionService), can also used to gather more information about the properties of the transaction service, by obtaining the following attributes:

- **transactionLogDirectory** The directory into which the log file(s) used for transaction service are placed
- **totalTranLifetimeTimeout** The total lifetime of the transaction (in seconds) before the container rolls it back. This value applies to Container Managed Transaction (CMT) beans only
- **asyncResponseTimeout** The total lifetime of the transaction (in seconds) before the container rolls it back
- **enableFileLocking** Enables the use of file locks when opening the transaction service recovery log
- **enableProtocolSecurity** Causes transaction service protocol messages to be sent securely
- **maximumTransactionTimeout** The maximum transaction timeout allowed for imported transactions. This value applies to Container Managed Transaction (CMT) beans, Bean Managed Transaction (BMT) beans, and imported transactions
- **clientInactivityTimeout** The number of seconds a transaction can remain inactive before it is rolled back
- **heuristicRetryLimit** The maximum number of times to retry transaction completion
- **heuristicRetryWait** The number of seconds to wait between retrying transaction completion
- **httpProxyPrefix** The HTTP prefix for WS-Transaction port SOAP addresses
- **httpsProxyPrefix** The HTTPS prefix for WS-Transaction port SOAP addresses
- **propogatedOrBMTTranLifetimeTimeout** The number of seconds a transaction can remain inactive before it is rolled back
- **LPSHeuristicCompletion** The transaction completion action to be taken when the outcome is unknown

The Transaction MBean, see API documentation - Application programming interfaces (package: Public MBean Interfaces, class: Transaction), can be used to commit, rollback, finish or remove from the list of heuristically completed transactions depending on the transaction's state, by invoking one of the following methods:

- commit: Heuristically commits the transaction
- rollback: Heuristically rolls back the transaction
- finish: Abandons retrying resources for the transaction
- removeHeuristic: Clears the transaction from the list

The Transaction MBean, see API documentation - Application programming interfaces (package: Public MBean Interfaces, class: Transaction), can also be used to gather more information about a transaction, by invoking the following methods:

- getPrintableStatus: Return the transaction status
- getGlobalTranName: Get the global identifier for the transaction
- listResources: List the resources for the transaction

The following script is an example of how to use the TransactionService MBean and Transaction MBean. The script should be run only against an application server, and not against the deployment manager or node agent.

Example

Working with manual transactions: example jacl script.

```
# get the TransactionService MBean
set servicembean [$AdminControl queryNames type=TransactionService,*]

# get the Transaction MBean
set mbean [$AdminControl queryNames type=Transaction,*]

set input 0
while {$input >= 0} {
        # invoke the listManualTransactions method
        set tranManualList [$AdminControl invoke $servicembean listManualTransactions]

        if {[llength $tranManualList] > 0} {
                puts "----Manual Transaction details---------------"
                set index 0
                foreach tran $tranManualList {
                        puts "   Index= $index tran= $tran"
                        incr index
                }
                puts "----End of Manual Transactions ---------------"
                puts "Select index of transaction to commit/rollback:"
                set input [gets stdin]
                if {$input < 0} {
                        puts "No index selected, exiting."
                } else {
                        set tran [lindex $tranManualList $input]
                        set commaPos [expr [string first "," $tran ]-1]
                        set localTID [string range $tran 0 $commaPos]
                        puts "Enter c to commit or r to rollback Transaction $localTID"
                        set input [gets stdin]
                        if {$input=="c"} {
                                puts "Committing transaction=$localTID"
                                $AdminControl invoke $mbean commit $localTID
                        }
                        if {$input=="r"} {
                                puts "Rolling back transaction=$localTID"
                                $AdminControl invoke $mbean rollback $localTID
                        }
                }
        } else {
                puts "No Manual transactions found, exiting"
                set input -1
```

```
        }
        puts " "

}
```

Working with manual transactions: example Jython script.

```
import sys
def wsadminToList(inStr):
        outList=[]
        if (len(inStr)>0 and inStr[0]=='[' and inStr[-1]==']'):
                tmpList = inStr[1:-1].split(" ")
        else:
                tmpList = inStr.split("\n")  #splits for Windows or Linux
        for item in tmpList:
                item = item.rstrip();        #removes any Windows "\r"
                if (len(item)>0):
                        outList.append(item)
        return outList
#endDef

servicembean = AdminControl.queryNames("type=TransactionService,*" )
mbean = AdminControl.queryNames("type=Transaction,*" )
input = 0

while (input >= 0):
        tranList = wsadminToList(AdminControl.invoke(servicembean, "listManualTransactions" ))

        tranLength = len(tranList)
        if (tranLength > 0):
                print "----Manual Transaction details---------------"
                index = 0
                for tran in tranList:
                        print "   Index=" , index , " tran=" , tran
                        index = index+1
                #endFor
                print "----End of Manual Transactions ---------------"
                print "Select index of transaction to commit/rollback:"
                input = sys.stdin.readline().strip()
                if (input == ""):
                        print "No index selected, exiting."
                        input = -1
                else:
                        tran = tranList[int(input)]
                        commaPos = (tran.find(",") -1)
                        localTID = tran[0:commaPos+1]
                        print "Enter c to commit or r to rollback transaction ", localTID
                        input = sys.stdin.readline().strip()
                        if (input == "c"):
                                print "Committing transaction=", localTID
                                AdminControl.invoke(mbean, "commit", localTID )
                        #endIf
                        elif (input == "r"):
                                print "Rolling back transaction=", localTID
                                AdminControl.invoke(mbean, "rollback", localTID )
                        #endIf
                        else:
                                input = -1
                        #endelse
                #endElse
        else:
                print "No transactions found, exiting"
                input = -1
        #endElse
        print " "

#endWhile
```

Managing transaction logging for optimum server availability

This topic describes how to manage transaction logging to optimize the availability of your application servers.

About this task

The transaction service writes information to the transaction log for every global transaction that involves two or more resources, or that is distributed across multiple servers. The transaction log is stored on disk and is used by the transaction service for recovery after a system or server crash. The transaction log for each application server consists of multiple subdirectories and files held in a single directory. To change the directory that an application server uses to store the transaction log, change the transaction log directory in the transaction service settings.

When a global transaction is completed, the information in the transaction log is no longer required, and the information is marked for deletion. The redundant information is garbage collected and intervals, and the space is reused by new transactions. The log files are created with a fixed size at server startup, so no further disk space allocation is required during the lifetime of the server.

If all the log space is in use when a transaction needs to save information, the transaction is rolled back and the message `CWWTR0083W: The transaction log is full. Transaction rolled back.` is reported to the system error log. No more transactions can commit until more log space is made available when existing active transactions complete.

The default disk space allocation for the transaction logs is 1M. For global transactions that involve only XA resources and that are either local to an application server or are distributed between enterprise beans running in remote application servers, the default disk space allocation is suitable for peak workloads of up to 4000 concurrent two-phase commit transactions. For global transactions that involve Web Services Atomic Transaction (WS-AT) transactions or interoperable OTS transactions, the default disk space allocation is suitable for peak workloads of up to 250 concurrent two-phase commit transactions. For higher workloads, consider using a larger transaction log. To change the disk space allocation for the transaction log files, change the transaction log directory in the transaction service settings.

You can monitor the number of concurrent global transactions by using the performance monitoring counters for transactions. The “Global transaction commit time” counter is a measure of how long a transaction takes to complete and, therefore, how long the log is in use by a transaction. If this value is high, then transactions are taking a long time to complete, which can be due to resource manager or network failures. If you ensure that this value is low, the log is more efficiently used and unlikely to become full.

Use the following tasks to manage transaction logging to optimize the availability of your application servers:

- “Configuring transaction aspects of servers for optimum availability.” Use this task to configure the transaction properties for an application server to help transactions to complete or recover more quickly.
- “Moving a transaction log from one server to another” on page 2162 Use this task if you need to move a transaction log between servers.
- “Restarting an application server on a different host” on page 2163 Use this task to restart the application server if you move a transaction log between hosts.
- “Configuring transaction properties for an application server” on page 2143 Use this task to change the directory or the disk space allocation for the transaction log files.

Configuring transaction aspects of servers for optimum availability

This topic describes some considerations and actions that you can take to configure transaction-related aspects of application servers, to optimize the availability of those servers.

About this task

This helps your transactions to complete or recover more quickly. After changing transaction-related properties of an application server, you must restart the server.

To configure transaction-related aspects of application servers for optimum availability, complete the following steps:

1. Store the transaction log files on a fast disk in a highly-available file system, such as a RAID device. The transaction log may need to be accessed by every global transaction and be used for transaction recovery after a crash. Therefore, the disk the log files are being written to should be on a highly-available file system, such as a RAID device.

   The performance of the disk also directly affects the transaction performance. In general, a global transaction makes two disk writes, one after the prepare phase when the outcome of the transaction is known (this information is forced to disk) and a further disk write at transaction completion. Therefore, the transaction logs should be placed on the fastest disks available and not make use of network mounted devices.
2. Mirror the transaction log files by using hardware disk mirroring or dual-ported disks. If log files have been mirrored or can be recovered, they can be used when restarting a failed server or moved to an another machine and another server started there to perform recovery.

   Hardware disk mirroring or dual-ported disks can be used by specifying the appropriate file system directory for the transaction logs using the WebSphere Administrative Console.
3. Specify the optimum location of the transaction log directory for application servers.

   By default, an application server places transaction log files in a subdirectory of the installed WebSphere Application Server, where the subdirectory name is the same as the server name.

   For example, the default directory for an application server named `server1` is

   `/QIBM/UserData/WebSphere/AppServer/`*`was_version`*`/Base/profiles/`*`profile_name`*`/tranlog/server1`

   where *was_version* indicates the version number for this installation of IBM WebSphere Application Server. For example `V6` for WebSphere Application Server Version 6.

   You can define a specific location for the transaction log directory for an application server by setting the **Transaction Log Directory** property for the server. If the directory for the transaction logs has not been created at application server start up, the directory structure is created for you.

   Note: If you change the transaction log directory, you should apply the change and restart the application server as soon as possible, to minimize the risk of problems occurring before the application server is restarted. For example, if a problem causes the server to fail (with in-flight transactions), the server next starts with the new log directory and is unable to automatically resolve in-flight transactions that were recorded in the old log directory.
4. Never allow more than one application server to concurrently use the same set of log files. Because the transaction logs record the state of global transactions within a server, if the logs become lost or corrupt, then transactions that are in the prepared state before failure can leave resources in an in-doubt state and prevent further updates or access to the resources by other users or servers. These transactions may need to be manually resolved by either committing or rolling back the transactions at the affected resource managers. The failed server can then be cold-started, which creates new empty transaction logs.

   If log files have been mirrored or can be recovered, they can be used when restarting the failed server or moved to an alternate server or machine and another server restarted to perform recovery, as described in the related tasks.

   Never allow more than one application server to concurrently use the same set of log files, because each server will destroy the information recorded by the other, resulting in corrupt log files that are unusable for future recovery purposes.
5. Configure application servers to always use the same listening port address at each startup. If you are running distributed transactions between multiple application servers, for example non-WebSphere EJB or Corba servers, the remote object references saved in the transaction log need to be redirected to the originating server on recovery.

   You must handle the redirection of remote object references so that transaction recovery can complete. For example, you must do this if an application server is deployed on WebSphere Application Server and runs distributed transactions with non-WebSphere EJB or Corba servers.

In particular, the default restart action of an application server is to use a different listening port address to the port when the server shuts down. This prevents transaction recovery from completing. To overcome this, you must configure application servers to always use the same listening port address at each startup (see the ORB property com.ibm.CORBA.ListenerPort in ORB service settings that can be added to the administrative console). You might need to make similar configuration changes to other application servers involved in transactions, so that you can access those servers during recovery.

Moving a transaction log from one server to another

This topic describes some considerations and actions that you can take to move the transaction logs for an application server to another server.

About this task

To move transaction logs from one application server to another, consider the following steps:

1. Move all the transaction log files for the application server. The transaction log directory for each server contains a number of files and subdirectories. When moving transaction logs from one server to another you must move all of the files and subdirectories together as a single unit; otherwise recovery may not complete resulting in data inconsistency.
2. For a server configuration where there are no distributed transactions, move the transaction logs to any server that has access to the same resource managers. For a single server or network-deployed server configuration where it is known there are no distributed transactions present in the logs, the transaction logs can be moved to any server (on any node) that has access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   If the server is in a different cell from the original server, you need to ensure that there is a JAAS alias available to the server that was used by the original server for accessing XA resources. In this case you should use wsadmin to construct the aliases, because if you use the administrative console to create the alias, then the node name gets prefixed to the alias.

   All the transaction log files for the original server need to be moved to a directory accessible by the new server. This can be accomplished by either renaming the transaction log directory or copying all the contents to the new server's transaction log directory before starting the new server.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).
3. For a network-deployed server configuration where there are distributed transactions, move the transaction logs to a server that has the same name and host IP address, and access to the same resource managers. For a network-deployed server configuration, when it is known there are distributed transactions present in the logs, there are more restrictions. Distributed transactions that access multiple servers log information about each server involved in the transaction. This information includes the server name and the IP address of the machine on which the server is running. When recovery is taking place on server restart, the server uses this information to contact the distributed servers and similarly, the distributed servers try to contact the server with the same original name. So, if a server fails and the logs need to the recovered on an alternative server, that alternative server needs to have the same name and host IP address as the original server. The alternative server also needs to have access to the same resource managers as the original server. For example, the server needs communication and valid security access to databases or message queues.

   Note: All servers within a cell must have unique names.

   Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue

to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

Restarting an application server on a different host

This topic describes some considerations and actions that you can take with transaction logs to restart an application server on a different host.

About this task

Moving transactions logs to a different host is similar to moving logs from one server to another, as described in Moving transaction logs from one server to another.

This involves moving an original application server on one host to an alternative server, which has access to the same resource managers, on another host. For a network-deployed server configuration, the alternative server must have the same name and host IP address as the original server.

Note: To complete transaction recovery, the application server uses the resource manager configuration information in the transaction logs. However, for the application server to continue to do new work with the same resource managers, the server must have an appropriate resource manager configuration (as for the original server).

To restart an application server on a different host, complete the following steps:

1. Ensure that the alternative application server is stopped.
2. Move all the transaction logs for the original server to the alternative application server, according to the considerations described in Moving transaction logs from one server to another.
3. Restart the alternative application server.

Interoperating transactionally between application servers

You can configure application servers so that transaction messages are sent and received between application servers at different versions of WebSphere Application Server. Depending on the version of the application server, you can set system properties, or use the transaction coordination authorization setting.

About this task

The transaction manager in WebSphere Application Server supports transactional interoperation with other transaction managers through either the CORBA Object Transaction Service (OTS) protocol, or, for JSR-109 compliant requests, the Web Services Atomic Transaction (WS-AT) protocol. Also, the transaction manager can coordinate XA resource managers and be coordinated by Java EE Connector Architecture 1.5 resource adapters.

- To interoperate transactionally, using the OTS protocol, to send requests from application servers that are WebSphere Application Server Version 5.0.2 or earlier to application servers that are Version 6 or later, you need to set the following system properties on application servers that are Version 5.0.2 or earlier.

```
com.ibm.ejs.jts.jts.ControlSet.nativeOnly=false
com.ibm.ejs.jts.jts.ControlSet.interoperabilityOnly=true
```

  For example, to send requests from application servers that are WebSphere Application Server Version 4.0.n to application servers that are WebSphere Application Server Version 6, you need to set the system properties on the Version 4.0.n application servers.

  You do not need to set these properties to receive requests on application servers that are WebSphere Application Server Version 5.0.2 or earlier from application servers that are Version 6 or later.

- When administrative security is enabled for application servers at WebSphere Application Server Version 6.0.2 or later, for such servers to interoperate transactionally with application servers at earlier versions, or with non-WebSphere Application Server servers, you must disable transaction coordination

authorization on the server. The transaction coordination authorization setting controls only the transaction protocol messages between servers that are used to coordinate the completion of a transaction. It does not affect application messages or the security of the server. When transaction coordination authorization is enabled, the server verifies that the sending server is authorized to handle prepare, commit, rollback, and one-phase commit messages.

To disable transaction coordination authorization on a server, use the following steps.

1. In the administrative console, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Container Settings] Container Services** → **Transaction Service**.
2. Clear the **Enable transaction coordination authorization** check box.
3. Click **Apply** or **OK**.
4. Save your changes to the master configuration.
5. Restart the server.

Chapter 14. Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

Your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application.

ActivitySessions

Configuring the default ActivitySession timeout for an application server

Use this task to configure the default ActivitySession timeout for an application server, after which any started ActivitySessions are completed automatically by the ActivitySession service.

About this task

The ActivitySession timeout is used to reset any ActivitySession whose remote client has failed to complete the ActivitySession in a timely fashion. The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the setSessionTimeout method of the UserActivitySession interface. If an ActivitySession that contains a transaction reaches the timeout, the transaction's timeout is accelerated so that it is timed out (and rolled back) immediately before the ActivitySession is reset.

To configure the default ActivitySession timeout for an application server, use the WebSphere Administrative console to complete the following steps:

1. Start the WebSphere Administrative console.
2. In the navigation pane, click **Servers** → **Application Servers** This displays a list of the application servers in the content pane.
3. In the Content pane, click the name of the application server that you want to configure. This displays the properties for the application server in the content pane.
4. Under Container Settings, click **Business Process Services** → **ActivitySession Service** This displays the ActivitySession service properties in the content pane.
5. Ensure that the **Enable service at server startup** check box is selected. You must enable the service for the timeout to have an effect.
6. Set the **ActivitySession timeout** property to the default timeout as an integer number of seconds.
   - -1 indicates that ActivitySessions never timeout
   - 0 indicates that the default timeout, 300 seconds, applies
   - Other values are an integer number of seconds

7. Click **OK**.
8. Save your changes to the master configuration.
9. To have the changed configuration take effect, stop then restart the application server.

Related concepts

The ActivitySession service
The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

ActivitySession service settings

Use this page to configure the properties of the ActivitySession service. The ActivitySession service is a unit-of-work service to coordinate one-phase resources or to extend the activation and passivation of an enterprise bean.

To view this administrative console page, click **Servers** → **Application servers** → *server_name* → **[Container Settings] Business Process Services** → **ActivitySession service**.

Enable service at server startup:

Specifies whether the application server attempts to start the ActivitySession service when the server next starts up.

Default Cleared

Range

Cleared
The server does not try to start the ActivitySession service. If ActivitySessions are to be used in applications that run on this server, the system administrator must select this property then restart the server.

Selected
When the application server starts, it attempts to start the ActivitySession service automatically.

Default timeout:

Specifies the default timeout for an activity session. A server automatically completes an activity session if a remote client has failed to complete the activity session within this time period.

The initial default timeout can be configured separately for each application server, and can be overridden programmatically by the UserActivitySession interface (setSessionTimeout).

Data type Integer
Units Seconds
Default 300 (5 minutes)
Range -1 through 1000000000 seconds
- -1 indicates that ActivitySessions never timeout
- 0 indicates that the default timeout applies
- Other values are an integer number of seconds

Disabling or enabling the ActivitySession service

Use this task to disable or enable the ActivitySession service for an application server.

About this task

You can use the ActivitySession **Startup** property to specify whether or not the ActivitySession service is started automatically for an application server.

To disable or enable the ActivitySession service for an application server, use the Administrative console to configure the ActivitySession **Startup** property:

1. Start the Administrative console.
2. In the navigation pane, click **Servers** → **Application Servers** This displays a list of the application servers in the content pane.
3. In the Content pane, click the name of the application server that you want to configure. This displays the properties for the application server in the content pane.
4. Under Container Settings, click **Business Process Services** → **ActivitySession Service** This displays the ActivitySession service properties in the content pane.
5. Select or clear the **Startup** property as needed:
   Selected
   The ActivitySession service is started when the application server is started. This enables applications that specify use of ActivitySessions in their deployment descriptors to run on such an application server.
   Cleared
   [Default] The ActivitySession service is not started when the application server is started. Applications that specify use of ActivitySessions in their deployment descriptors cannot start on such an application server.

   Any attempt to start an application that uses ActivitySessions is rejected and a message issued:

   ```
   WACS0043E: Error found starting an application.  application_name specified an ActivitySession
   attribute that is not allowed when the ActivitySession service is not enabled
   ```

   If this happens during server startup, the server continues to start without the application.
6. Click **OK**.
7. Save your changes to the master configuration.
8. To have the changed configuration take effect, stop then restart the application server.

Related concepts

The ActivitySession service
The ActivitySession service provides an alternative unit-of-work (UOW) scope to that provided by global transaction contexts. An ActivitySession context can be longer-lived than a global transaction context and can encapsulate global transactions.

Related reference

“ActivitySession service settings” on page 2166
Use this page to configure the properties of the ActivitySession service. The ActivitySession service is a unit-of-work service to coordinate one-phase resources or to extend the activation and passivation of an enterprise bean.

Application profiling

Task overview: Application profiling

You can use application profiling to configure multiple access intent policies on the same entity bean.

About this task

Application profiling reflects the fact that different units of work have different use patterns for enlisted entities and can require different kinds of support from the server runtime environment. For more information, see Application Profiling.

1. Assembling applications for application profiles. This topic describes how to configure tasks, create application profiles, and configure tasks on profiles.
2. Managing application profiles. This topic describes how to add and remove tasks from application profiles using the administrative console.
3. Using the TaskNameManager API. This topic describes how to programmatically set the current task name, but you should use this technique sparingly. Wherever possible, use the declarative method instead, which results in more portable function.

Application profiling

You can use application profiling to identify particular units of work to the product runtime environment. The run time can tailor its support to the exact requirements of that unit of work.

Application profiling requires accurate knowledge of an application's transactional configuration and the interaction of the application with its persistent state during the course of each transaction.

You can execute the analysis in either closed world or open world mode. A closed-world analysis assumes that all possible clients of the application are included in the analysis and that the resulting analysis is complete and correct. The results of a closed-world analysis report the set of all transactions that can be invoked by a web, JMS, or application client. The results exclude many potential transactions that never execute at run time.

An open-world analysis assumes that not all clients are available for analysis or that the analysis cannot return complete or accurate results. An open-world analysis returns the complete set of possible transactions.

The results of an analysis persist as an application profiling configuration. The assembly tool establishes container managed tasks for servlets, JavaServer Pages (JSP) files, application clients, and Message Driven Beans (MDBs). Application profiles for the tasks are constructed with the appropriate access intent for the entities enlisted in the transaction represented by the task. However, in practice, there are many situations where the tool returns at best incomplete results. Not all applications are amenable to static analysis. Some factory and command patterns make it impossible to determine the call graphs. The tool does not support the analysis of *ActivitySessions*.

You should examine the results of the analysis very carefully. In many cases you must manually modify them to meet the requirements of the application. However, the tool can be an effective starting place for most applications and may offer a complete and quick configuration of application profiles for some applications.

Access intent is the only runtime component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two new concepts in order to achieve this function: *tasks* and *profiles*.

Tasks A task is a configurable name for a unit of work. *Unit of work* in this case means either a transaction or an ActivitySession. The task name is typically assigned declaratively on a J2EE component that can initiate a unit of work. Most commonly, the task is configured on a method of an Enterprise JavaBeans file that is declared either for container-managed transactions or bean-managed transactions. Any unit of work that begins in the scope of a configured task is associated with that task name. A unit of work can only be named when it is initiated, and the name cannot change for the lifetime of that unit of work. A unit of work ignores any subsequent

task name configurations at any point after it has begun. The task is used for the duration of its unit of work to identify configured policies specific to that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Profiles

A profile is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a CMR getter, or a dynamic query) requires data to be retrieved from the back end system, the current task associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent. If a request is operating in the absence of a task, the runtime environment uses either a method-level access intent (if any) or a bean-level default access intent.

Note: The application profile configuration is application scope configuration data. If any Enterprise JavaBean (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

Tasks and units of work considerations:

The application profiling function works under the unit of work (UOW) concept. UOW in this case means either a transaction or an ActivitySession.

The task name on a method is used only when a UOW is begun, because of that method being invoked. This gives it a more predictable data access pattern based on the active unit of work. To be more specific, this approach ensures that a bean type with only one configured access intent is loaded within a UOW,

because a bean is configured with only one access intent within an application profile. This configured access intent for a bean type is determined at assembly time and is enforced by the Application Profile service.

A task name is always associated with a unit of work, and that task name does not change for the duration of that UOW. When a UOW associated with a method is begun because of that method being invoked, if a task name is associated with the method then that task name is used to name the UOW. A task assigned to a unit of work is considered a named UOW.

If a task name is not associated with the method that began the UOW, then a default access intent is used and the UOW is unnamed. A unit of work can only be named when the UOW is begun and that task name remains for the life of the UOW. Furthermore, the task assigned to a UOW can never be changed for the life of that UOW. Any task names associated with a method are ignored if that method does not begin a UOW (either container managed or component managed).

It is not possible to change the task name assigned to a unit of work. However, it is possible that in a call sequence consisting of many different application calls a different task name might need to be used for different calls. In this case it is important for the deployer to begin a new UOW and associate with the UOW the necessary task name. For example, assume you have the following beans: sb1 is a session bean, eb2 and eb3 are container managed persistence (CMP) entity beans. When sb1 is called, a transaction is begun and task 't1' is associated with it. Further assume that sb1 then calls eb2 and eb3. If neither eb2 or eb3 create a unit of work, then these beans execute within the UOW context from sb1 and as such its task name (t1). If eb2 or eb3 need to execute within a task name other than t1, then these beans must define a unit of work and associate with it the appropriate task name.

Note that if an application deployer does not specifically configure a transaction on a method, WebSphere Application Server creates a global transaction by default. This is important because if a task is defined on a method, but a UOW is not specifically configured on that method, the EJB container automatically creates a global transaction on behalf of that method. As such, this task name is associated with the UOW and any application profiles mapped to this task are used.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

Application profiles:

An application profile is the set of access intent policies that should be selectively applied for a particular unit of work (a transaction or *ActivitySession*).

Application profiling enables applications to run under different sets of policies depending on the active task under which the application is operating.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the

active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Consider an application that centralizes the student records for a school district. These records are frequently accessed by the school district's central office in order to generate reports. The report generation process would be optimized if it held no locks with the back end system, and if the records could be read into memory with as few back end operations as possible. Occasionally, however, the records are updated by the students' instructors. Without the ability to distinguish between transactions, the developer is forced to assume a worst-case scenario and, wishing to use pessimistic concurrency, lock the records for all transactions.

Using the application profiling service, the developer can configure in as many ways as necessary the access intent under which the students' records are loaded. Under one profile, the records can be configured with an exclusive pessimistic update intent, not only locking-out competing transactions but ensuring that the student is not removed from the system before the transaction completes. Under another profile, the records can be configured with an optimistic intent as part of an object graph that is read from the back end system in a single database operation. The task represented by the pessimistic profile receives the strong-locking semantics required for certain transactions, while the task represented by the optimistic profile receives the performance benefits appropriate for other transactions.

Application profiling performance considerations:

Application profiling enables assembly configuration techniques that improve your application run time, performance and scalability. You can configure tasks that identify incoming requests, identify access intents determining concurrency and other data access characteristics, and profiles that map the tasks to the access intents.

The capability to configure the application server can improve performance, efficiency and scalability, while reducing development and maintenance costs. The application profiling service has no tuning parameters, other than a checkbox for disabling the service if the service is not necessary. However, the overhead for the application profile service is small and should not be disabled, or unpredictable results can occur.

Access intents enable you to specify data access characteristics. The WebSphere runtime environment uses these hints to optimize the access to the data, by setting the appropriate isolation level and concurrency. Various access intent hints can be grouped together in an access intent policy.

In the product, it is recommended that you configure bean level access intent for loading a given bean. Application profiling enables you to configure multiple access intent policies on the entity bean, if desired. Some callers can load a bean with the intent to read data, while others can load the bean for update. The capability to configure the application server can improve performance, efficiency, and scalability, while reducing development and maintenance costs.

Access intents enable the EJB container to be configured providing optimal performance based on the specific type of enterprise bean used. Various access intent hints can be specified declaratively at deployment time to indicate to WebSphere resources, such as the container and persistence manager, to provide the appropriate access intent services for every EJB request.

The application profiling service improves overall entity bean performance and throughput by fine tuning the run time behavior. The application profiling service enables EJB optimizations to be customized for multiple user access patterns without resorting to ″worst case″ choices, such as pessimistic update on a bean accessed with the findByPrimaryKey method, regardless of whether the client needs it for read or for an update.

Application profiling provides the capability to define the following hierarchy: **Container-Managed Tasks** > **Application Profiles** > **Access Intent Policies** > **Access Intent Overrides**. Container-managed tasks identify units of work (UOW) and are associated with a method or a set of methods. When a method associated with the task is invoked, the task name is propagated with the request. For example, a UOW refers to a unique path within the application that can correspond to a transaction or ActivitySession. The name of the task is assigned declaratively to a Java EE client or servlet, or to the method of an enterprise bean. The task name identifies the starting point of a call graph or subgraph; the task name flows from the starting point of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the configured task. As a best practice, wherever a UOW starts, for example, a transaction or an ActivitySession, assign a task to that starting point.

The application profile service associates the propagated tasks with access intent policies. When a bean is loaded and data is retrieved, the characteristics used for the retrieval of the data are dictated by the application profile. The application profile configures the access intent policy and the overrides that should be used to access data for a specific task.

Access intent policies determine how beans are loaded for specific tasks and how data is accessed during the transaction. The access intent policy is a named group of access intent hints. The hints can be used, depending on the characteristics of the database and resource manager. Various access intent hints applied to the data access operation govern data integrity. The general rule is, the more data integrity, the more overhead. More overhead causes lower throughput and the opportunity for simultaneous data access from multiple clients.

If specified, access intent overrides provide further configuration for the access intent policy.

Best practices

Application profiling is effective in a variety of different scenarios including:

- **The same bean is loaded with different data access patterns**

  The same bean or set of beans can be reused across applications, but each of those applications has differing requirements for the bean or for beans within the invocation graph. One application can require that beans be loaded for update, while another application requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.
- **Different clients have different data access requirements**

  The same bean or set of beans can be used for different types of client requests. When those clients have different requirements for the bean, or for beans within the invocation graph, application profiling can be used to tailor the bean loading characteristics to the requirements of the client. One client can require beans be loaded for update, while another client requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.

Monitoring tools

You can use the Tivoli Performance Viewer, database and logs as monitoring tools.

You can use the Tivoli Performance Viewer to monitor various metrics associated with beans in an application profiling configuration. The following sections describe at a high level the Tivoli Performance Viewer metrics that reflect changes when access intents and application profiling are used:

- **Collection scope**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor this information to determine the difference between using the ActivitySession scope versus the transaction scope. For the transaction scope, depending on how the container transactions are defined, activates and passivates can be associated with method invocations. The application could use the ActivitySession scope to reduce the frequency of activates and passivates. For more information, see ″Using the ActivitySession service.″
- **Collection increment**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular findByPrimaryKey operation. For example, if the collection increment is set to 10, rather than the default 25, the *Num Activates* value shows 25 for the initial findByPrimaryKey, before any result set iterator runs. If the number of activates rarely exceeds the collection increment, consider reducing the collection increment setting.
- **Resource manager prefetch increment**

  The resource manager prefetch increment is a hint acted upon by the database engine to depend upon the database. The Tivoli Performance Viewer does not have a metric available to show the effect of the resource manager prefetch increment setting.
- **Read ahead hint**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular request. If a read ahead association is not in use, the *Num Activates* value shows a lower initial number. If a read ahead association is in use, the *Num Activates* value represents the number of activates for the entire call graph.

Database tools are helpful in monitoring the different bean loading characteristics that introduce contention and concurrency issues. These issues can be solved by application profiling, or can be made worse by the misapplication of access intent policies.

Database tools are useful for monitoring locking and contention characteristics, such as locks, deadlocks and connections open. For example, for locks the DB2 Snapshot Monitor can show statistics for lock waits, lock time-outs and lock escalations. If excessive lock waits and time-outs are occurring, application profiling can define specific client tasks that require a more string level of locking, and other client tasks that do not require locking. Or, a different access intent policy with less restrictive locking could be applied. After applying this configuration change, the snapshot monitor shows less locking behavior. Refer to information about the database you are using on how to monitor for locking and contention.

The **application server logs** can be monitored for information about rollbacks, deadlocks, and other data access or transaction characteristics that can degrade performance or cause the application to fail.

Application profiling tasks

Tasks are named units of work. They are the mechanism by which the runtime environment determines which access intent policies to apply when an entity bean's data is loaded from the back end system.

Application profiles enable developers to configure an entity bean with multiple access intent policies; if there are *n* instances of profiles in a given application, each bean can be configured with as many as *n* access intent policies.

A task is associated with a transaction or an ActivitySession at the initiation of the unit of work. The task, which cannot change for the lifetime of the unit of work, is always available anywhere within the scope of that unit of work to apply the access intent policy configured for that particular unit of work.

If an enterprise application is configured to use application profiling in any part of the application, then application profiling is active and method-level access intent configurations are ignored when units of works are associated with known-to-application tasks.

If an entity bean is loaded in a unit of work that is not associated with a task, or is associated with a task that is unassociated with an application profile, the default bean-level access intent or the method-level access intent configuration is applied. If a unit of work is associated with a task that is configured with an application profile, the bean-level access intent configuration within the appropriate application profile is applied.

Note: The application profile configuration is application scope configuration data. If any enterprise Javabean (EJB) module contains an application profile configuration, all other EJB modules are implicitly regulated by the Application Profiling service even if they do not contain application profile configuration data.

For example, an application has two EJB modules: EJBModule1 and EJBModule2.

The EJBModule1 has an application profile named AppProfile1. This AppProfile1 is registered by a task named task1. This task1 becomes a *known-to-application task* and is honored when associated with a unit of work within this application. With the presence of any known-to-application task, method level access intent configurations are ignored and only bean level access intent configurations are applied.

The EJBModule2 contains no application profile configuration data. All entity beans are **not** configured with bean level access intent explicitly, but some methods have method level access intent configurations. If an entity bean in the EJBModule2 is loaded in a unit of work that is associated with task1, the bean-level access intent configuration is applied and method level access intent configuration is ignored. Because the bean level access intent is not set explicitly, the default bean level access intent, which is wsPessimisticUpdate-WeakestLockAtLoad, is applied.

The active task depends upon the current unit of work mechanism. If the current unit of work is a global transaction, then the task is the name associated with that transaction. If the global transaction was not named when it was initiated, then there is no active task anywhere in the scope of that transaction.

If the current unit of work is a local transaction associated with an ActivitySession, then the task is the name associated with that ActivitySession. If the ActivitySession was not named when it was initiated, then there is no active task for any local transaction bound to that ActivitySession. If the current unit of work is a local transaction that is not associated with an ActivitySession, then the task is the name associated with that local transaction. If the local transaction was not associated with a task when the local transaction was initiated, then there is no active task for the duration of that local transaction. In other words, the active task is the task associated with the unit of work on the thread that is coordinating database resources. If the controlling unit of work was not associated with a task when that unit of work was initiated, then there is no active task in the scope of that unit of work.

For example, consider a school district application that calls through a session bean in order to interact with student records. One method on the session bean allows administrators to modify the students' records; another method supports student requests to view their own records. Without application profiling, the two tasks would operate anonymously and the runtime environment would be unable to distinguish work operating on behalf of one task or the other. To optimize the application, a developer can configure one of the methods on the session bean with the task ″updateRecords″ and the other method on the session bean with the task ″readRecords″. When registered with an application profile that has the student bean configured with the appropriate locking access intent, the ″updateRecords″ task is assured that it is

not unnecessarily blocking transactions that need to only read the records. For more information about the relationships between tasks and units of work, see “Tasks and units of work considerations” on page 2169.

Tasks can be configured to be managed by the container or to be programmatically established by the application. Container managed tasks can be configured on servlets, JavaServer Pages (JSP) files, application clients, and the methods of Enterprise JavaBeans (EJB). Configured container-managed tasks are only associated with units of work that the container initiates after the task name is set. Application managed tasks can be configured on all J2EE components. In the case of enterprise beans they must be bean managed transactions.″

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service’s console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

Application profiling interoperability

Using application profiling with 5.x compatibility mode or in a clustered environment with mixed product versions and mixed platforms can affect its behavior in different ways.

The effect of 5.x Compatibility Mode

Application profiling supports *forward* compatibility. Application profiles created in previous versions of WebSphere Application Server (Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1) can only run in WebSphere Application Server Version 6 or later if the Application Profiling 5.x Compatibility Mode attribute is turned on. If the 5.x Compatibility Mode attribute is off, Version 5 application profiles might display unexpected behavior.

Similarly, application profiles that you create using the latest version of WebSphere Application Server are not compatible with Version 5 or earlier versions. Even applications configured with application profiles run on Version 6.x servers with the Application Profiling 5.x Compatibility Mode attribute turned on cannot interact with applications configured with profiles run on Version 5 servers.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service’s console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.x client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to **true** in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the *launchClient* command.

WebSphere Application Server Enterprise Edition Version 5.0.2

If you use WebSphere Application Server Enterprise Edition 5.0.2, you must apply WebSphere Application Server Version 5 service pack 7 or later service pack to enable Application Profiling interoperability.

Managing application profiles

Using the administrative console, you can add tasks to or remove tasks from application profiles.

1. Start the administrative console.
2. Select **Applications > Enterprise Applications >** *application_name* **> Application Profiles >** *profile_name* **> Tasks**.
3. On the Tasks collection page, you can add new tasks to the profile, delete tasks, edit current task settings, and so on.

   Note that, within the scope of an application, no task can be configured on more than one application profile. In such a situation, your application cannot be restarted until you correct the configuration.
4. Save your configuration.
5. Restart the application in order for your changes to take effect.

Using the TaskNameManager interface

Using the TaskNameManager interface, you can programmatically set the current task name. It enables both overriding of the current task associated with the thread of execution and resetting of the current task with the original task.

About this task

Except for J2EE 1.3 applications that are running on a server where the 5.x Compatibility Mode attribute is selected, this interface cannot be used within Enterprise JavaBeans that are configured for container-managed transactions or container-managed ActivitySessions because units of work can only be associated with a task at the exact time that the unit of work is initiated. The call to set the task name must therefore be invoked before the unit of work is begun. Units of work cannot be named after they are begun. Calls on this interface during the execution of a container-managed unit of work are simply ignored.

Application profiling does not support queries of the task that is in operation at run time. Instead, applications interact with logical task names that are declaratively configured as application managed tasks. Logical references enable the actual task name to be changed without having to recompile applications.

Wherever possible, avoid setting tasks programmatically. The declarative method results in more portable function that can be easily adjusted without requiring redevelopment and recompilation.

Note: If you select the 5.x Compatibility Mode attribute on the Application Profile Service's console page, then tasks configured on J2EE 1.3 applications are not necessarily associated with units of work and can arbitrarily be applied and overridden. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. Tasks are not communicated on requests between applications that are running under the Application Profiling 5.x Compatibility Mode and applications that are not running under the compatibility mode.

For a Version 6.0 client to interact with applications run under the Application Profiling 5.x Compatibility Mode, you must set the *appprofileCompatibility* system property to *true* in the client process. You can do this by specifying the *-CCDappprofileCompatibility=true* option when invoking the launchClient command.

1. Configure application-managed tasks. Application profiling requires that a task name reference be declared for any task that is to be set programmatically. Task name references introduce a level of indirection so that the actual task set at run time can be adjusted by reassembly without requiring recoding or recompilation. Any attempt to set a task name that is undeclared as a task reference results in the raising of an exception. If a unit of work has already begun when a task name is set, then that existing unit of work is not associated with the task name. Only units of work that are begun after the task name is set are associated with the task.

   Configure application-managed tasks as described in the following topics. To complete these tasks see the assembly tool information center:

- Configure application managed tasks for web components.
- Configure application managed tasks for application clients.
- Configure application managed tasks for Enterprise JavaBeans.

2. Perform a Java Naming and Directory Interface (JNDI) lookup on the TaskNameManager interface:

```
InitialContext ic = new InitialContext();
TaskNameManager tnManager = ic.lookup
("java:comp/websphere/AppProfile/TaskNameManager");
```

   The *TaskNameManager* interface is not bound into the namespace if the application profiling service is disabled.

3. Set the task name:

```
try {
tnManager.setTaskName("updateAccount");
}
catch (IllegalTaskNameException e) {
// task name reference not configured. Handle error.
}
// . . .
//
```

   The name passed to the setTaskName() method (″updateAccount″ in this example) is actually a task name reference that you configured in step one.

4. Begin a UserTransaction

   Note: If you are using a J2EE 1.3 application with the 5.x Compatibility mode set, the task name set in step 3 is now the active task name and you can disregard this step.

   If you are using a J2EE application and the compatibility mode is not set, or if you are using a J2EE 1.4 application, you must begin a transaction for the task name to become active. A task name can only be associated with a transaction. Furthermore, it is associated with a transaction when that transaction is begun, and that task name is associated with the transaction for the life of the transaction. Therefore, the task name set above is not active at this point. You must begin a UserTransaction as the following code snippet illustrates:

```
try{
    InitialContext initCtx = new InitialContext();
    userTran = (UserTransaction) initCtx.lookup("java:comp/UserTransaction");
    userTran.begin();
}
catch(Exception e){
}
// . . .
//
```

   Notice the resetTaskName() method on the TaskNameManager interface. Resetting the task name has no effect unless called by a J2EE 1.3 application running on a server for which the 5.x Compatibility Mode attribute is selected on the Application Profile Service's console page. This is not a recommended mode of operation and can lead to unexpected deadlocks during database access. A call to resetTask( ) should only be used by J2EE 1.3 applications when the 5.x Compatibility mode is set to undo the effects of any setTaskName() method operations and reestablish whatever task name was current when the component began execution. If the setTaskName() method has not been called, the resetTaskName() method has no effect.

TaskNameManager interface:

The TaskNameManager is the programmatic interface to the application profiling function. Because on rare occasions it may be necessary to programmatically set the current task name, the TaskNameManager interface enables both overriding of the current task associated with the thread of execution and resetting of the current task with the original task.

Application profiling enables you to identify particular units of work to the WebSphere Application Server runtime environment. The run time can tailor its support to the exact requirements of that unit of work. Access intent is currently the only runtime component that makes use of the application profiling functionality. For example, you can configure one transaction to load an entity bean with strong update locks and configure another transaction to load the same entity bean without locks.

Application profiling introduces two concepts in order to achieve this function: tasks and profiles.

A *task* is a configurable name for a unit of work. *Unit of work* in this case means either a transaction or an ActivitySession.

A *profile* is simply a mapping of a task to a set of access intent policies that are configured on entity beans. When an invocation on a bean (whether by a finder method, a container managed relationship (CMR) getter, or a dynamic query) requires data to be retrieved from the back end system, the task of the active unit of work associated with the request is used to determine the exact requirement of the transaction. The same bean loads and behaves differently in the context of the task-to-profile mapping. Each profile provides the developer an opportunity to reconfigure the application's access intent.

Except for J2EE 1.3 applications that are executing on a server where the *5.x Compatibility Mode* attribute is selected, this interface cannot be used within Enterprise JavaBeans that are configured for container-managed transactions or container-managed ActivitySessions because units of work can only be associated with a task at the exact time that the unit of work is initiated. The call to set the task name must therefore be started before the unit of work is begun. Units of work cannot be named after they are begun. Calls on this interface during the execution of a container-managed unit of work are simply ignored.

The TaskNameManager interface is available to all J2EE components using the following Java Naming and Directory Interface (JNDI) lookup:

```
java:comp/websphere/AppProfile/TaskNameManager
```

```
package com.ibm.websphere.appprofile;

/**
* The TaskNameManager is the programmatic interface
* to the application profiling function. Using this interface,
* programmers can set the current task name on the
* thread of execution. The task name must have been
* configured in the deployment descriptors as a task
* reference associated with a task. The set task
* name's scope is the duration of the method
* invocation in the EJB and Web components and for
* the duration of the client process, or until the
* resetTaskName() method is invoked.
*/
public interface TaskNameManager {

/**
* Set the thread's current task name to the specified
* parameter. The task name must have been configured as
* a task reference with a corresponding task or the
* IllegalTaskName exception is thrown.
*/
public void setTaskName(String taskName) throws IllegalTaskNameException;

/**
* Sets the thread's task name to the value that was set
* at, or imported into, the beginning of the method
* invocation (for EJB and Web components) or process
* (for J2EE clients).
*/
public void resetTaskName();

}
```

Application profiling exceptions

The following exceptions are thrown in response to various illegal actions related to application profiling.

com.ibm.ws.exception.RuntimeWarning

This exception is thrown when the application is started, if the application is configured incorrectly. The startup is consequently terminated. Some examples of bad configurations include:

- A task configured on two different application profiles.
- A method configured with two different task run-as policies .

com.ibm.websphere.appprofile.IllegalTaskNameException

This exception is raised if an application attempts to programmatically set a task when that task has not been configured as a task name reference.

Asynchronous beans

Using asynchronous beans

The asynchronous beans feature adds a new set of APIs that enable Java 2 Platform Enterprise Edition J2EE applications to run asynchronously inside an Integration Server.

About this task

This topic provides a brief overview of the tasks involved in using asynchronous beans. For a more detailed description of the asynchronous beans model, review the conceptual topic Asynchronous beans. For detailed information on the programming model for supported asynchronous beans interfaces, see the topic Work managers.

1. Configure work managers.
2. Configure timer managers.
3. Assemble applications that use asynchronous beans work managers.
4. Develop work objects to run code in parallel.
5. Develop event listeners.
6. Develop asynchronous scopes.

Asynchronous beans

An asynchronous bean is a Java object or enterprise bean that can run asynchronously by a Java Platform, Enterprise Edition (Java EE) application, using the Java EE context of the asynchronous bean creator.

Asynchronous beans can improve performance by enabling a Java EE program to decompose operations into parallel tasks. Asynchronous beans support the construction of stateful, active Java EE applications. These applications address a segment of the application space that Java EE has not previously addressed (that is, advanced applications that require application threading, active agents within a server application, or distributed monitoring capabilities).

Asynchronous beans can run using the Java EE security context of the creator Java EE component. These beans also can run with copies of other Java EE contexts, such as:

- Internationalization context
- Application profiles, which are not supported for Java EE 1.4 applications and deprecated for Java EE 1.3 applications
- Work areas

Asynchronous bean interfaces

Four types of asynchronous beans exist:

Work object

There are two work interfaces that essentially accomplish the same goal. The legacy

Asynchronous Beans work interface is com.ibm.websphere.asynchbeans.Work, and the CommonJ work interface is commonj.work.Work. A work object runs parallel to its caller using the work manager `startWork` or `schedule` method (`startWork` for legacy Asynchronous Beans and `schedule` for CommonJ). Applications implement work objects to run code blocks asynchronously. For more information on the Work interface, see the generated API documentation.

Timer listener

This interface is an object that implements the commonj\timers\TimerListener interface. Timer listeners are called when a high-speed transient timer expires. For more information on the TimerListener interface, see the generated API documentation.

Alarm listener

An alarm listener is an object that implements the com.ibm.websphere.asynchbeans.AlarmListener interface. Alarm listeners are called when a high-speed transient alarm expires. For more information on the AlarmListener interface, see the generated API documentation.

Event listener

An event listener can implement any interface. An event listener is a lightweight, asynchronous notification mechanism for asynchronous events within a single Java virtual machine (JVM). An event listener typically enables Java EE components within a single application to notify each other about various asynchronous events.

Supporting interfaces

Work manager

Work managers are thread pools that administrators create for Java EE applications. The administrator specifies the properties of the thread pool and a policy that determines which Java EE contexts the asynchronous bean inherits.

CommonJ Work manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a Java EE 1.4 environment, each JNDI lookup of a work manager does not return a new instance of the WorkManager. All the JNDI lookup of work managers within a scope have the same instance.

Timer manager

Timer managers implement the commonj.timers.TimerManager interface, which enables Java EE applications, including servlets, EJB applications, and JCA Resource Adapters, to schedule future timer notifications and receive timer notifications. The timer manager for Application Servers specification provides an application-server supported alternative to using the J2SE `java.util.Timer` class, which is inappropriate for managed environments.

Event source

An event source implements the com.ibm.websphere.asynchbeans.EventSource interface. An event source is a system-provided object that supports a generic, type-safe asynchronous notification server within a single JVM. The event source enables event listener objects, which implement any interface to be registered. For more information on the EventSource interface, see the generated API documentation.

Event source events

Every event source can generate its own events, such as listener count changed. An application can register an event listener object that implements the class com.ibm.websphere.asynchbeans.EventSourceEvents. This action enables the application to catch events such as listeners being added or removed, or a listener throwing an unexpected exception. For more information on the EventSourceEvents class, see the generated API documentation.

Additional interfaces, including alarms and subsystem monitors, are introduced in the topic Developing Asynchronous scopes, which discusses some of the advanced applications of asynchronous beans.

Transactions

Every asynchronous bean method is called using its own transaction, much like container-managed transactions in typical enterprise beans. It is very similar to the situation when an Enterprise Java Beans

(EJB) method is called with TX_NOT_SUPPORTED. The runtime starts a local transaction before invoking the method. The asynchronous bean method is free to start its own global transaction if this transaction is possible for the calling Java EE component. For example, if an enterprise bean creates the component, the method that creates the asynchronous bean must be TX_BEAN_MANAGED.

When you call an entity bean from within an asynchronous bean, for example, you must have a global transactional context available on the current thread. Because asynchronous bean objects start local transactional contexts, you can encapsulate all entity bean logic in a session bean that has a method marked as TX_REQUIRES or equivalent. This process establishes a global transactional context from which you can access one or more entity bean methods.

If the asynchronous bean method throws an exception, any local transactions are rolled back. If the method returns normally, any incomplete local transactions are completed according to the unresolved action policy configured for the bean. EJB methods can configure this policy using their deployment descriptor. If the asynchronous bean method starts its own global transaction and does not commit this global transaction, the transaction is rolled back when the method returns.

Access to Java EE component metadata

If an asynchronous bean is a Java EE component, such as a session bean, its own metadata is active when a method is called. If an asynchronous bean is a simple Java object, the Java EE component metadata of the creating component is available to the bean. Like its creator, the asynchronous bean can look up the java:comp namespace. This look up enables the bean to access connection factories and enterprise beans, just as it would if it were any other Java EE component. The environment properties of the creating component also are available to the asynchronous bean.

The `java:comp` namespace is identical to the one available for the creating component; the same restrictions apply. For example, if the enterprise bean or servlet has an EJB reference of `java:comp/env/ejb/MyEJB`, this EJB reference is available to the asynchronous bean. In addition, all of the connection factories use the same resource-sharing scope as the creating component.

Connection management

An asynchronous bean method can use the connections that its creating Java EE component obtained using java:comp resource references. (For more information on resource references, see References). However, the bean method must access those connections using a get, use or close pattern. There is no connection caching between method calls on an asynchronous bean. The connection factories or datasources can be cached, but the connections must be retrieved on every method call, used, and then closed. While the asynchronous bean method can look up connection factories using a global Java Naming and Directory Interface (JNDI) name, this is not recommended for the following reasons:

- The JNDI name is hard coded in the application (for example, as a property or string literal).
- The connection factories are not shared because there is no way to specify a sharing scope.

For code examples that demonstrate both the correct and the incorrect ways to access connections from asynchronous bean methods, see the topic Example: Asynchronous bean connection management.

Deferred start of Asynchronous Beans

Asynchronous beans support deferred start by allowing serialization of Java EE service context information. The `WorkWithExecutionContext createWorkWithExecutionContext(Work r)` method on the WorkManager interface will create a snapshot of the Java EE service contexts enabled on the WorkManager. The resulting `WorkWithExecutionContext` object can then be serialized and stored in a database or file. This is useful when it is necessary to store Java EE service contexts such as the current security identity or Locale and later inflate them and run some work within this context. The `WorkWithExecutionContext` object can run using the startWork() and doWork() methods on the WorkManager interface.

All `WorkWithExecutionContext` objects must be deserialized by the same application that serialized it. All EJBs and classes must be present in order for Java to successfully inflate the objects contained within.

Deferred start and security

The asynchronous beans security service context might require Common Secure Interoperability Version 2 (CSIv2) identity assertion to be enabled. Identity assertion is required when a `WorkWithExecutionContext` object is deserialized and run to Java Authentication and Authorization Service (JAAS) subject identity credential assignment. Review the following topics to better understand if you need to enable identity assertion, when using a `WorkWithExecutionContext` object:

- Configuring Common Secure Interoperability Version 2 and Security Authentication Service authentication protocol
- Identity Assertion

There are also issues with interoperating with `WorkWithExecutionContext` objects from different versions of the product. See Interoperating with asynchronous beans .

JPA-related limitations

Use of asynchronous beans within a JPA extended persistence context is not supported.

A JPA extended persistence context is inconsistent with the scheduling and multi-threading capabilities of asynchronous beans and will not be accessible from an asynchronous bean thread.

Likewise, an asynchronous bean should not be created such that it takes a javax.persistence.EntityManager (or subclass) as a parameter since EntityManager instances are not intended to be thread safe.

Work managers:

A work manager is a thread pool created for Java Platform, Enterprise Edition (Java EE) applications that use asynchronous beans.

Using the administrative console, an administrator can configure any number of work managers. The administrator specifies the properties of the work manager, including the Java EE context inheritance policy for any asynchronous beans that use the work manager. The administrator binds each work manager to a unique place in Java Naming and Directory Interface (JNDI). You can use work manager objects in any one of the following interfaces:

- Asynchronous beans
- CommonJ work manager (For details, see the CommonJ work manager section in this article.)

The selected type of interface is resolved during the JNDI lookup time. The interface type is the value that you specify in the ResourceRef, rather than the interface type specified in the configuration object. For example, you can have one ResourceRef for each interface per configuration object, and each ResourceRef lookup returns that appropriate type of instance.

The work managers provide a programming model for the Java EE 1.4 applications. For more information, see the Programming model section in this article.

When writing a Web or Enterprise JavaBeans (EJB) component that uses asynchronous beans, the developer should include a resource reference in each component that needs access to a work manager. For more information on resource references, see the article References. The component looks up a work manager using a logical name in the component, java:comp namespace, just as it looks up a data source, enterprise bean or connection factory.

The deployer binds physical work managers to logical work managers when the application is deployed.

For example, if a developer needs three thread pools to partition work between bronze, silver, and gold levels, the developer writes the component to pick a logical pool based on an attribute in the client application profile. The deployer has the flexibility to decide how to map this request for three thread pools. The deployer might decide to use a single thread pool on a small machine. In this case, the deployer binds all three resource references to the same work manager instance (that is, the same JNDI name). A larger machine might support three thread pools, so the deployer binds each resource reference to a different work manager. Work managers can be shared between multiple Java EE applications installed on the same server.

An application developer can use as many logical work managers as necessary. The deployer chooses whether to map one physical work manager or several to the logical work manager defined in the application.

All Java EE components that need to share asynchronous scope objects must use the same work manager. These scope objects have an affinity with a single work manager. An application that uses asynchronous scopes should verify that all of the components using scope objects use the same work manager.

When multiple work managers are defined, the underlying thread pools are created in a Java virtual machine (JVM) only if an application within that JVM looks up the work manager. For example, there might be ten thread pools (work managers) defined, but none are actually created until an application looks these pools up.

Note: Asynchronous beans do not support submitting work to remote JVMs.

CommonJ Work Manager

The CommonJ work manager is similar to the work manager. The difference between the two is that the CommonJ work manager contains a subset of the asynchronous beans work manager methods. Although CommonJ work manager functions in a Java EE 1.4 environment, the interface does not return a new instance for each JNDI naming lookup, since this specification is not included in the Java EE specification.

Remote start of work. The CommonJ Work specification optional feature for work running remotely is not supported. Even if a unit of work implements the `java.io.Serializable` interface, the unit of work does not run remotely.

How to look up a work manager

An application can look up a work manager as follows. Here, the component contains a resource reference named `wm/myWorkManager`, which was bound to a physical work manager when the component was deployed:

```
InitialContext ic = new InitialContext();
WorkManager wm = (WorkManager)ic.lookup("java:comp/env/wm/myWorkManager");
```

Inheritance Java EE contexts

Asynchronous beans can inherit the following Java EE contexts.

Internationalization context
: When this option is selected and the internationalization service is enabled, and the internationalization context that exists on the scheduling thread is available on the target thread.

Work area
: When this option is selected, the work area context for every work area partition that exists on the scheduling thread is available on the target thread.

Application profile (deprecated)
: Application profile context is not supported and not available for Java EE 1.4 applications. For Java EE 1.3 applications, when this option is selected, the application profile service is enabled,

and the application profile service property, **5.x compatibility mode**, is selected. The application profile task that is associated with the scheduling thread is available on the target thread for Java EE 1.3 applications. For Java EE 1.4 applications, the application profile task is a property of its associated unit of work, rather than a thread. This option has no effect on the behavior of the task in Java EE 1.4 applications. The scheduled work that runs in a Java EE 1.4 application does not receive the application profiling task of the scheduling thread.

Security

The asynchronous bean can be run as anonymous or as the client authenticated on the thread that created it. This behavior is useful because the asynchronous bean can do only what the caller can do. This action is more useful than a RUN_AS mechanism, for example, which prevents this kind of behavior. When you select the **Security** option, the JAAS subject that exists on the scheduling thread is available on the target thread. If not selected, the thread runs anonymously.

Component metadata

Component metadata is relevant only when the asynchronous bean is a simple Java object. If the bean is a Java EE component, such as an enterprise bean, the component metadata is active.

The contexts that can be inherited depend on the work manager used by the application that creates the asynchronous bean. Using the administrative console, the administrator defines the sticky context policy of a work manager by selecting the services on which the work manager is to be made available.

Programming model

Work managers support the following programming models.

- **CommonJ Specification**. The Application Server Version 6.0 CommonJ programming model uses the WorkManager and TimerManager to manage threads and timers asynchronously in the Java EE 1.4 environment.
- **Asynchronous beans and CommonJ specification extensions**. The current asynchronous beans Event Source, asynchronous scopes, subsystem monitors and Java EE Context interfaces are a part of the CommonJ extension.

The following table describes the method mapping between the CommonJ and Asynchronous beans APIs. You can change the current asynchronous beans interfaces to use the CommonJ interface, while maintaining the same functions.

CommonJ package	API	Asynchronous beans package	API
Work manager		Work manager	
Asynchronous beans	Field - IMMEDIATE (long)		Field - IMMEDIATE (int)
	Field - INDEFINITE		Field - INDEFINITE
	schedule(Work) throws WorkException, IllegalArgumentException		startWork(Work) throws WorkException, IllegalArgumentException
	schedule(Work, WorkListener) throws WorkException, IllegalArgumentException **Note:** Configure the work manager work timeout property to the value you previously specified as `timeout_ms` on `startWork`. The default timeout value is INDEFINITE.		startWork(Work, timeout_ms, WorkListener) throws WorkException, IllegalArgumentException

	waitForAll(workItems, timeout_ms)		join(workItems, JOIN_AND, timeout_ms)
	waitForAny(workItems, timeout_ms)		join(workItems, JOIN_OR, timeout_ms)
WorkItem		WorkItem	
	getResult		getResult
	getStatus		getStatus
WorkListener		WorkListener	
	workAccepted(WorkEvent)		workAccepted(WorkEvent)
	workCompleted(WorkEvent)		workCompleted(WorkEvent)
	workRejected(WorkEvent)		workRejected(WorkEvent)
	workStarted(WorkEvent)		workStarted(WorkEvent)
WorkEvent		WorkEvent	
	Field - WORK_ACCEPTED		Field - WORK_ACCEPTED
	Field - WORK_COMPLETED		Field - WORK_COMPLETED
	Field - WORK_REJECTED		Field - WORK_REJECTED
	Field - WORK_STARTED		Field - WORK_STARTED
	getException		getException
	getType		getType
	getWorkItem().getResult() **Note:** This API is valid only after the work is complete.		getWork
Work	(extends Runnable)	Work	(Extends Runnable)
	isDaemon		*
	release		release
RemoteWorkItem	Not in this release. Use Distributed WorkManager in Extended Deployment or future release	NA	
TimerManager		AlarmManager	
	resume		*
	schedule(Listener, Date)		create(Listener, context, time) ** need to convert the parameters
	schedule(Listener, Date, period)		
	schedule(Listener, delay, period)		
	scheduleAtFixedRate(Listener, Date, period)		
	scheduleAtFixedRate(Listener, delay, period)		
	stop		
	suspend		
Timer		Alarm	
	cancel		cancel

	getPeriod		
	getTimerListener		getAlarmListener
	scheduledExecutionTime		
TimerListener		AlarmListener	
	timerExpired(timer)		fired(alarm)
StopTimerListener		Not applicable	
	timerStop(timer)		
CancelTimerListener		Not applicable	
	timerCancel(timer)		
WorkException	(Extends Exception)	WorkException	(Extends WsException)
WorkCompletedException	(Extends WorkException)	WorkCompletedException	(Extends WorkException)
WorkRejectedException	(Extends WorkException)	WorkRejectedException	(Extends WorkException)

For more information on work manager APIs, refer to the Javadoc.

Work manager examples

Table 47. Look up work manager

Asynchronous beans	CommonJ

`InitialContext ctx = new InitialContext();`
`com.ibm.websphere.asynchbeans.WorkManager wm =`
`(com.ibm.websphere.asynchbeans.WorkManager)`
`ctx.lookup("java:comp/env/wm/MyWorkMgr");`
	`InitialContext ctx = new InitialContext();`
	`commonj.work.WorkManager wm =`
	`(commonj.work.WorkManager)`
	`ctx.lookup("java:comp/env/wm/MyWorkMgr");`

Table 48. Create your work using MyWork

Asynchronous beans	CommonJ

`public class MyWork implements`
`com.ibm.websphere.asynchbeans.Work {`
`public void release() {`
`.....`
`}`
`public void run() {`
`System.out.println("Running.....");`
`}`
	`public class MyWork implements`
	`commonj.work.Work{`
	`public boolean isDaemon() {`
	`return false;`
	`}`
	`public void release () {`
	`.....`
	`}`
	`public void run () {`
	`System.out.println("Running.....");`
	`}`

Table 49. Submit the work

Asynchronous beans	CommonJ

Table 49. Submit the work (continued)

```
 MyWork work1 = new MyWork(new URI =
"http://www.example./com/1");
 MyWork work2 = new MyWork(new URI =
"http://www.example./com/2");

 WorkItem item1;
 WorkItem item2;
 Item1=wm.startWork(work1);
 Item2=wm.startWork(work2);

 // case 1: block until all items are done
 ArrayList col1 = new ArrayList();
 Col1.add(item1);
 Col1.add(item2);
 wm.join(col1, WorkManager.JOIN_AND,
 (long)WorkManager.IMMEDIATE);
 // when the works are done
 System.out.println("work1 data="+work1.getData());
 System.out.println("work2 data="+work2.getData());

 // case 2: wait for any of the items to complete.
 Boolean ret = wm.join(col1,
  WorkManager.JOIN_OR, 1000);
```

```
 MyWork work1 = new MyWork(new URI =
"http://www.example./com/1");
 MyWork work2 = new MyWork(new URI =
"http://www.example./com/2");

 WorkItem item1;
 WorkItem item2;
 Item1=wm.schedule(work1 );
 Item2=wm.schedule(work2);

 // case 1: block until all items are done
 Collection col1 = new ArrayList();
 col1.add(item1);
 col1.add(item2);
 wm.waitForAll(col1, WorkManager.IMMEDIATE);
 // when the works are done
 System.out.println("work1 data="+work1.getData());
 System.out.println("work2 data="+work2.getData());


 // case 2: wait for any of the items to complete.
 Collection finished = wm.waitForAny(col1,
 // check the workItems status
 if (finished != null) {
   Iterator I = finished.iterator();
   if (i.hasNext()) {
      WorkItem wi = (WorkItem) i.next();
       if (wi.equals(item1)) {
         System.out.println("work1 =
      "+ work1.getData());
       } else if (wi.equals(item2)) {
         System.out.println("work1 =
      "+ work1.getData());
       }
   }
 }
```

Table 50. Create a timer manager

Asynchronous beans	CommonJ

`InitialContext ctx  = new InitialContext();`
`com.ibm.websphere.asynchbeans.WorkManager wm  =`
`(com.ibm.websphere.asynchbeans.WorkManager)`
`ctx.lookup("java:comp/env/wm/MyWorkMgr");`
 	
`AsynchScope ascope;`
`Try {`
`Ascope = wm.createAsynchScope("ABScope");`
`} Catch (DuplicateKeyException ex)`
`{`
`Ascope = wm.findAsynchScope("ABScope");`
`ex.printStackTrace();`
`}`
 	
`// get an AlarmManager`
`AlarmManager  aMgr= ascope.getAlarmManager();`
	`InitialContext ctx  = new InitialContext();`
	`Commonj.timers.TimerManager tm =`
	`(commonj.timers.TimerManager)`
	`ctx.lookup("java:comp/env/tm/MyTimerManager");`

Table 51. Fire the timer

Asynchronous beans	CommonJ

Table 51. Fire the timer (continued)

`// create alarm`
`ABAlarmListener listener = new ABAlarmListener();`
`Alarm am =`
`  aMgr.create(listener, "SomeContext", 1000*60);`
	`// create Timer`
	`TimerListener listener =`
	`new StockQuoteTimerListener("qqq",`
	`"johndoe@example.com");`
	`Timer timer = tm.schedule(listener, 1000*60);`
	
	`// Fixed-delay: schedule timer to expire in`
	`// 60 seconds from now and repeat every`
	`// hour thereafter.`
	`Timer timer = tm.schedule(listener, 1000*60,`
	`1000*30);`
	
	`// Fixed-rate: schedule timer to expire in`
	`// 60 seconds from now and repeat every`
	`// hour thereafter`
	`Timer timer = tm.scheduleAtFixedRate(listener,`
	`1000*60, 1000*30);`

Timer managers:

The timer manager combines the functions of the asynchronous beans alarm manager and asynchronous scope. So, when a timer manager is created, it internally uses an asynchronous scope to provide the timer manager life cycle functions.

You can look up the timer manager in the Java Naming and Directory Interface (JNDI) name space. This capability is different from the alarm manager that is retrieved through the asynchronous beans scope. Each lookup of the timer manager returns a new logical timer manager that can be destroyed independently of all other timer managers.

A timer manager can be configured with a number of thread pools through the administrative console. For deployment you can bind this timer manager to a resource reference at assembly time, so the resource reference can be used by the application to look up the timer manager.

The Java code to look up the timer manager is:

```
InitialContext ic = new InitialContext();
TimerManager tm = (TimerManager)ic.lookup("java:comp/env/tm/TimerManager");
```

The programming model for setting up the alarm listener and the timer listener is different. The following code example shows that difference.

Table 52. Set up the timer listener

Asynchronous beans	**CommonJ**

Table 52. Set up the timer listener (continued)

public class ABAlarmListener implements AlarmListener {
public void fired(Alarm alarm) {
System.out.println("Alarm fired. Context =" + alarm.getContext());
}
	public class StockQuoteTimerListener implements TimerListener {
	String context;
	String url;
	public StockQuoteTimerListener(String context,
	String url){
	this.context = context;
	This.url = url;
	}
	public void timerExpired(Timer timer) {
	System.out.println("Timer fired. Context ="+ ((StockQuoteTimerListener)timer.getTimerListener()) .getContext());
	
	}
	public String getContext() {
	return context;
	}
	}

Example: Using connections with asynchronous beans:

An asynchronous bean method can use the connections that its creating Java Platform, Enterprise Edition (Java EE) component obtained using java:comp resource references.

For more information on resource references, see the topic References. The following is an example of an asynchronous bean that uses connections correctly:

```
class GoodAsynchBean
{
 DataSource ds;
 public GoodAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource
  // as class instance data.
  InitialContext ic = new InitialContext();
  // it is assumed that the created Java EE component has this
  // resource reference defined in its deployment descriptor.
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
 }
 // When the asynchronous bean method is called, get a connection,
 //  use it, then close it.
 void anEventListener()
 {
  Connection c = null;
  try
  {
   c = ds.getConnection();
   // use the connection now...
  }
  finally
  {
   if(c != null) c.close();
  }
 }
}
```

The following example of an asynchronous bean that uses connections incorrectly:

```
class BadAsynchBean
{
 DataSource ds;
 // Do not do this. You cannot cache connections across asynch method calls.
```

```
 Connection c;

 public BadAsynchBean()
  throws NamingException
 {
  // ok to cache a connection factory or datasource as
  // class instance data.
  InitialContext ic = new InitialContext();
  ds = (DataSource)ic.lookup("java:comp/env/jdbc/myDataSource");
  // here, you broke the rules...
  c = ds.getConnection();
 }
 // Now when the asynch method is called, illegally use the cached connection
 // and you likely see J2C related exceptions at run time.
 // close it.
 void someAsynchMethod()
 {
  // use the connection now...
 }
}
```

Work manager service settings

Use this page to enable or disable the work manager service that manages work manager resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Work Manager Service** .

Startup:

Specifies whether the server attempts to start the work manager service.

Default Selected

Range

Selected
When the application server starts, it attempts to start the work manager service automatically.

Cleared
The server does not try to start the work manager service. If work manager resources are to be used on this server, the system administrator must start the work manager service manually or select this property then restart the server.

Configuring timer managers

A timer manager acts as a thread pool for application components that use asynchronous beans. Use the administrative console to configure timer managers. The timer manager service is enabled by default.

Before you begin

If you are not familiar with timer managers, review the conceptual section, Timer managers, in the Asynchronous beans topic.

About this task

You can define multiple timer managers for each cell. Each timer manager is bound to a unique place in Java Naming and Directory Interface (JNDI).

Note: The timer manager service is only supported from within the Enterprise Java Beans (EJB) container or Web container. Looking up and using a configured timer manager from a J2EE application client container is not supported.

1. Start the administrative console.
2. Select **Resources > Asynchronous beans > Timer managers**.
3. Specify a **Scope** value and click **New**.
4. Specify the following required properties:
   Scope The scope of the configured resource. This value indicates the location for the configuration file.
   Name The display name for the timer manager.
   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the timer manager. This name is used by asynchronous beans that need to look up the timer manager. Each timer manager must have a unique JNDI name within the cell.
   Number of Timer Threads
   The maximum number of threads that are used for timers.
5. [Optional] Specify a **Description** and a **Category** for the timer manager.
6. [Optional] Select the **Service Names** (J2EE contexts) on which you want this timer manager to be made available. Any asynchronous beans that use this timer manager then inherit the selected J2EE contexts from the component that creates the bean. The list of selected services also is known as the ″sticky″ context policy for the timer manager. Selecting more services than are actually required might impede performance.
7. Save your configuration.

Results

The timer manager is now configured and ready for access by application components that need to manage the start of asynchronous code.

Timer manager collection

Use this page to view the configuration properties of timer managers, which enable applications to schedule future timer notifications and to receive timer notification callbacks to application-specified listeners within a Java 2 Platform, Enterprise Edition (J2EE) environment. The timer manager binds to the Java Naming and Directory Interface (JNDI) name space.

A timer manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources** > **Asynchronous beans** > **Timer managers**.

Name:

Specifies the name by which the timer manager is known for administrative purposes.

Data type String

JNDI Name:

Specifies the JNDI name used to look up the timer manager in the name space.

Data type String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies a description of this timer manager for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this timer manager.

Data type String

Timer manager settings:

Use this page to modify timer manager settings. Timer managers enable applications to schedule future timer notifications and to receive timer notification callbacks to application-specified listeners within a Java Platform, Enterprise Edition (Java EE) environment. The timer manager binds to the Java Naming and Directory Interface (JNDI) name space.

A timer manager contains a pool of threads bound into JNDI.

To view this administrative console page, click **Resources** > **Asynchronous beans** > **Timer managers** *timermanager_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which the timer manager is known for administrative purposes.

Data type String

JNDI name:

Specifies the JNDI name used to look up the timer manager in the namespace.

Data type String

Description:

Specifies a description of this timer manager for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this timer manager.

Data type String

Service names:

Specifies a list of services to make available to this timer manager.

Asynchronous beans can inherit Java EE context information by enabling one or more Java EE service contexts on the timer manager resource in the product administrative console or by setting the serviceNames attribute of the TimerManagerInfo configuration object. When specifying the serviceNames attribute each enabled service should be separated by a semicolon, for example, `security;UserWorkArea;com.ibm.ws.i18n`. When a Java EE service context is enabled, it propagates the context from the scheduling thread to the target thread. If not enabled, the target thread does not inherit the context of the scheduling thread and a default context is applied. Any related Java EE context that is already present on the thread is suspended before any new Java EE context is applied.

The context information of each selected service is propagated to each timer that is created using this timer manager. Selecting services that are not needed can negatively impact performance.

Work area
Use the administrative console or the `UserWorkArea` service name to enable work area partitions. When enabled, the work area context for every work area partition that exists on the scheduling thread is available on the target thread. This feature is optional.

Security
Use the administrative console or the `security` service name to enable the Java Authentication and Authorization Service (JAAS) subject. When this feature and administrative security are enabled, the JAAS subject that is present on the scheduling thread is applied to the target thread. If not enabled, the target thread is run anonymously without a JAAS subject on the thread. This feature is optional.

Internationalization
Use the administrative console or the `com.ibm.ws.i18n` service name to enable the internationalization context information. When the internationalization context and the Internationalization service is enabled, the internationalization context that exists on the scheduling thread is available on the target thread. This feature is optional.

Number of timer threads:

Specifies the maximum number of threads that are used for timers.

Data type Integer

Configuring work managers

A work manager acts as a thread pool for application components that use asynchronous beans. Use the administrative console to configure work managers.

Before you begin

If you are not familiar with work managers, review the conceptual topic, Work managers.

About this task

The work manager service is always enabled. In previous versions of the product, the work manager service could be disabled using the administration console or configuration service. The work manager service configuration objects are still present in the configuration service, but the enabled attribute is ignored.

You can define multiple work managers for each cell. Each work manager is bound to a unique place in Java Naming and Directory Interface (JNDI).

Note: The work manager service is only supported from within the Enterprise Java Beans (EJB) Container or Web Container. Looking up and using a configured work manager from a J2EE application client container is not supported.

1. Start the administrative console.
2. Select **Resources > Asynchronous beans > Work managers**.
3. Specify a **Scope** value and click **New**.
4. Specify the required properties for work manager settings.

   Scope The scope of the configured resource. This value indicates the location for the configuration file.

   Name The display name for the work manager.

   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the work manager. This name is used by asynchronous beans that need to look up the work manager. Each work manager must have a unique JNDI name within the cell.

   Number of Alarm Threads
   The maximum number of threads to use for processing alarms. A single thread is used to monitor pending alarms and dispatch them. An additional pool of threads is used for dispatching the threads. All alarm managers on the asynchronous beans associated with this work manager share this set of threads. A single alarm thread pool exists for each work manager, and all of the asynchronous beans associated with the work manager share this pool of threads.

   Minimum Number Of Threads
   The number of threads to be kept in the thread pool, created as needed.

   Maximum Number Of Threads
   The maximum number of threads to be created in the thread pool. The maximum number of threads can be exceeded temporarily if the **Growable** check box is selected. These additional threads are discarded when the work on the thread completes.

   Thread Priority
   The priority to assign to all threads in the thread pool.

   Every thread has a priority. Threads with higher priority are run before threads with lower priority. For more information about how thread priorities are used, see the javadoc for the setPriority method of the java.lang.Thread class in the Java Standard Edition specification.
5. [Optional] Specify a **Description** and a **Category** for the work manager.
6. [Optional] Select the **Service Names** (J2EE contexts) on which you want this work manager to be made available. Any asynchronous beans that use this work manager then inherit the selected J2EE contexts from the component that creates the bean. The list of selected services also is known as the ″sticky″ context policy for the work manager. Selecting more services than are actually required might impede performance.

   Other optional fields include:

   Work timeout
   Specifies the number of milliseconds to wait before a scheduled work object is released. If a value is not specified, then the timeout is disabled.

   Work request queue size
   Specifies the size of the work request queue. The work request queue is a buffer that holds scheduled work objects and may be any value 1 or greater. The thread pool pulls work from this queue. If you do not specify a value or the value is 0, the queue size is managed automatically. Large values can consume significant system resources.

   Work request queue full action
   Specifies the action taken when the thread pool is exhausted, and the work request queue is full. This action starts when you submit non-daemon work to the work manager. If set to FAIL, the work manager API methods creates an exception instead of blocking.
7. Save your configuration.

Results

The work manager is now configured and ready for access by application components that need to manage the start of asynchronous code.

Work manager collection

Use this page to view the collection properties of work managers, which contain a pool of threads bound into the Java Naming and Directory Interface.

To view this administrative console page, click **Resources > Asynchronous beans > Work managers**.

Name:

Specifies the name by which the work manager is known for administrative purposes.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name used to look up the work manager in the namespace.

Data type String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies the description of this work manager for administrative purposes.

Category:

Specifies a category name that is used to classify or group this work manager.

Work manager settings:

Use this page to modify work manager settings. Work managers contain a pool of threads that are bound into Java Naming and Directory Interface.

To view this administrative console page, click **Resources > Asynchronous beans > Work managers >** *workmanager_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which the work manager is known for administrative purposes.

JNDI Name:

Specifies the Java Naming and Directory Interface (JNDI) name used to look up the work manager in the namespace.

Description:

Specifies the description of this work manager for administrative purposes.

Category:

Specifies a string that you can use to classify or group this work manager.

Work timeout:

Specifies the number of milliseconds to wait before attempting to release a unit of work. The timeout interval begins when the unit of work starts, rather than when the unit of work is submitted.

Default 0

Work request queue size:

Specifies the size of the work request queue. The work request queue is a buffer that holds submitted units of work until a thread is available for them to run.

Default 0

Work request queue full action:

Specifies the action taken when the thread pool is exhausted, and the work request queue is full. This action starts when you submit non-daemon work to the work manager.

If set to FAIL, the work manager API methods creates an exception instead of blocking.

Default BLOCK
Range FAIL

Service names:

Specifies a list of services to make available to this work manager.

Asynchronous beans can inherit J2EE context information by enabling one or more J2EE service contexts on the work manager resource in the WebSphere administrative console or by setting the serviceNames attribute of the WorkManagerInfo configuration object. When specifying the serviceNames attribute each enabled service should be separated by a semicolon. For example: `security;UserWorkArea;com.ibm.ws.i18n`. When a J2EE service context is enabled, it propagates the context from the scheduling thread to the target thread. If not enabled, the target thread does not inherit the context of the scheduling thread and a default context is applied. Any related J2EE context that is already present on the thread is suspended before any new J2EE context is applied.

The context information of each selected service is propagated to each work or alarm that is created using this work manager. Selecting services that are not needed can negatively impact performance.

Application profile (deprecated)	Use the administrative console or the AppProfileService service name to enable the application profile tasks. Application profile context is not supported and not available for J2EE 1.4 applications. For J2EE 1.3 applications, the application profile context is deprecated and is only available when **Application Profile Service 5.x Compatibility Mode** is enabled and both the scheduling thread and target thread are J2EE 1.3 applications. When enabled, all application profile tasks that are available on the scheduling thread are available on the target thread. The scheduled work that runs in a J2EE 1.4 application does not get the application profiling task of the scheduling thread. This feature is optional.
Work area	Use the administrative console or the `UserWorkArea` service name to enable work area partitions. When enabled, the work area context for every work area partition that exists on the scheduling thread is available on the target thread. This feature is optional.
Security	Use the administrative console or the `security` service name to enable the Java Authentication and Authorization Service (JAAS) subject. When this feature and administrative security are enabled, the JAAS subject that is present on the scheduling thread is applied to the target thread. If not enabled, the target thread is run anonymously without a JAAS subject on the thread. This feature is optional.
Internationalization	Use the administrative console or the `com.ibm.ws.i18n` service name to enable the internationalization context information. When the internationalization context and the Internationalization service is enabled, the internationalization context that exists on the scheduling thread is available on the target thread. This feature is optional.

Thread pool properties:

Specifies the priority of the threads available in this work manager.

Number of alarm threads	Specifies the desired maximum number of threads used for alarms. The default value is 2.
Minimum number of threads	Specifies the minimum number of threads available in this work manager.
Maximum number of threads	Specifies the maximum number of threads available in this work manager.
Thread priority	Specifies the priority of the threads available in this work manager.
Growable	Specifies whether the number of threads in this work manager can be increased.

Dynamic cache

Task overview: Using the dynamic cache service to improve performance

Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance. WebSphere Application Server consolidates several caching activities including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, the results of a servlet after it runs, and metadata.

About this task

The dynamic cache service works within an application server Java virtual machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls through a servlet service method or a command

execute method, and either stores the output of the object to the cache or serves the content of the object from the dynamic cache.

1. Enable the dynamic cache service globally. To use the features associated with dynamic caching, you must enable the service in the administrative console. See “Using the dynamic cache service” on page 2210 for more information.
2. Configure the type of caching that you are using:
   - “Configuring servlet caching” on page 2215.
   - “Configuring portlet fragment caching” on page 2216.
   - “Configuring Edge Side Include caching” on page 2226.
   - “Configuring command caching” on page 2245.
   - “Example: Caching Web services” on page 2201.
   - “Configuring the Web services client cache” on page 2247.
3. Monitor the results of your configuration using the dynamic cache monitor. For more information, see “Displaying cache information” on page 2272.
4. If you have any problems with your configuration, see the *Troubleshooting and support* PDF.

What to do next

To use the DistributedMap and DistributedObjectCache interfaces for the dynamic cache, see “Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache” on page 2253.

Disk cache infrastructure enhancements

Several performance enhancements are available for the dynamic cache service.

The dynamic cache service supports persisting objects to disk (specified by a file system location) so that objects that are evicted from the memory cache are not regenerated by the application server. Objects are written to disk when they are evicted from memory using a Least Recently Used (LRU) eviction algorithm. The objects in the memory cache may also be flushed to disk on normal server shutdown. Java objects that need to be offloaded to the disk should be serializable.

The disk offload function includes the following functions:

- An internal disk cache format for faster deletions and support for new options to limit disk cache size
- The disk cache garbage collector, which evicts objects out of the cache when a configured high threshold is reached
- Four new performance modes to tune your disk cache performance:
  - High performance/memory usage mode - keeps all metadata in system memory and provides the highest performance
  - Balanced performance/memory usage mode - provides optimal balance of performance and memory usage by keeping some metadata in system memory
  - Custom performance/memory usage mode - allows explicit configuration of the memory usage and customization of performance requirements
  - Low performance/memory usage mode - stores most of the metadata on disk for users who are very constrained on system memory

Limiting the disk cache. The dynamic cache service provides mechanisms to limit the use of the disk cache by specifying the size of the disk cache in gigabytes, in addition to the maximum number of entries that are persisted to the disk. The disk cache is considered full when either of these limits is reached and forms the basis for eviction of objects from the disk. If the cache subsystem cannot offload any more data to disk, due to either an out-of-disk space condition, insufficient space on disk, or an exception when writing data to disk as a result of a possibly corrupt disk, the disk offload capability is disabled to prevent data integrity problems. The event is logged and the disk cache subsystem is deleted. This prevents serving corrupt data from the cache on a restart. If the option to persist cache data is turned on, some

information such as dependency and template information is flushed to disk on a server shutdown. If a disk full situation occurs during this shutdown process, any partially-persisted and un-persisted dependency or template data is removed from the cache. A side effect of this, to preserve integrity, is to invalidate the cached objects that are associated with the dependency or template data.

Disk cache size in GB. The disk cache size in GB option pertains primarily to the object data (which includes the cached object, its identifier, and metadata such as expiration time), template information and dependency information that are written to disk. The cache subsystem allocates separate storage and volumes (each of which can grow to 1 GB) for object data, templates and dependencies, as needed. When the total number of volumes on disk exceeds the specified cache size, any subsequent data that is written to disk is discarded until more space is made available by the disk cache garbage collector. To preserve data integrity, any information that is related to discarded objects is invalidated as well. The thresholds for garbage collection (described below) and the disk cache full state are associated with the space available for object data. It is also possible that in certain, rare scenarios, as information is flushed to disk, critical system data needs to be written to disk, which may cause the total file system space required to exceed up to 5% of the specified maximum limit. It is recommended that there be at least 25% of actual file system space available for disk caching over and above the specified disk cache size in GB. It is also required that each cache instance has a unique disk offload location and it is recommended that each offload location be on a dedicated disk partition. The cache file system employs a logical file manager to manage storage allocation for cached objects, therefore the file system size or the size of the files in the cache directory may not be an accurate gauge of the available space for the cache subsystem. At the same time, because of the adjusted limit, the cache subsystem may encounter a cache full state prior to the approaching the specified maximum limit as measured in allocated file system space. The PMI counters provide a better picture of how full the cache is.

Related concepts

“Eviction policies using the disk cache garbage collector”
The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

Related reference

“Dynamic cache PMI counter definitions” on page 2206
The dynamic cache statistics interface is defined as WSDynamicCacheStats under the `com\ibm\websphere\pmi\stat` package.

“Dynamic cache MBean statistics” on page 2203
The dynamic cache service provides an MBean interface to access cache statistics.

Eviction policies using the disk cache garbage collector

The disk cache garbage collector is responsible for evicting objects out of the disk cache, based on a specified eviction policy.

The garbage collector keeps a certain amount of space on disk available, which is governed by the configuration attribute that limits the amount of disk space that is used for caching objects. To enable the eviction policy, enable the Limit disk cache size in GB and/or Limit disk cache size in entries options in the administrative console.

The garbage collector is triggered when the disk space reaches a specified high threshold (a percentage of the Limit disk cache size in entries or in GB) and evicts objects, based on the eviction policy, from the disk in the background until the disk cache size reaches a specified low threshold (a percentage of the Limit disk cache size in entries or in GB). Eviction triggers when one or both of the high thresholds is reached for Limit disk cache size in GB and Limit disk cache size in entries. The supported policies are:

- None: This is the default policy. Objects are evicted only when they expire, or if they are invalidated.
- Random: The expired objects are removed first. If the disk size still has not reached the low threshold limit, objects are picked from the disk cache in random order and removed until the disk size reaches a low threshold limit.

- Size: The expired objects are removed first. If the disk size still has not reached the low threshold limit, then largest-sized objects are removed until the disk size reaches a low threshold limit.

Limit disk cache size in GB and High Threshold determines when to trigger eviction and when the disk cache is considered near full. It is computed as a function of the user-specified limit. If the specified limit is *10* GB (3 GB is the minimum), the cache subsystem initially creates three files that can grow to 1 GB in size for cache data, dependency ID information, and template information. Each time more space is needed to contain cache data, dependency ID information, or template information, a new file is created. Each of these files grow in 1 GB increments until the total number of files that are created is equal to disk cache in size in GB (in this case ten). Although the initial size of the new file may be much smaller than 1 GB, the dynamic cache service always rounds up to the next GB.

Eviction triggers when the cache data size reaches the high threshold and continues until the cache data size reaches the low threshold. Calculation of cache data size is dynamic. The following formula describes how to calculate the actual cache data size limit:

```
cache data size limit = disk cache size (in GB) - number of dependency files per GB - number of template files
```

When the cache data size limit is defined, the trigger point is calculated as follows:

```
eviction trigger point = cache data size limit * high threshold
size of evicted entries = cache data size * (high threshold - low threshold)
```

Consider the following scenarios:

- **Scenario 1**
  - Disk cache size in GB = 10 GB
  - High threshold = 90%
  - Low Threshold = 80%

  Initially, there is one file for dependency ID and template ID.

  ```
  cache data size limit = 10-(1+1) = 8 GB
  eviction trigger point = 8 * 90% = 7.2 GB
  size of evicted entries = 8 * (90% - 80%) = 0.8 GB
  ```

  In the above scenario, eviction starts when the data cache size reaches 7.2 GB and continues until the cache size is 6.4 GB (7.2 - 0.8).
- **Scenario 2**

  In scenario 1, if the dependency files grow to more than 1 GB, an additional dependency file generates. The eviction trigger point launches dynamically as follows:

  ```
  cache data size limit = 10 - (2+1) = 7GB
  eviction trigger point = 7 * 90% = 6.3GB
  size of evicted entries = 7 * (90% - 80%) = 0.7GB
  ```

  In the above scenario, eviction starts when the data cache size reaches 6.3 GB, and continues until the cache size in 5.6 GB (6.3 - 0.7).

Disk cache eviction for limit disk cache size in entries. Consider the following scenario:

- Disk cache size in entries = 100000
- High threshold = 90%
- Low threshold = 80%

```
eviction trigger point = 100000 * 90% = 90000
number of entries evicted = 100000 * (90% - 80%) = 10000
```

In this scenario, eviction starts when the number of cache entries reaches 90000 and 10000 entries are evicted from the cache.

Example: Caching Web services

This topic includes examples of building a set of cache policies and SOAP messages for a Web services application.

The following is a example of building a set of cache policies for a simple Web services application. The application in this example stores stock quotes and has operations to read, update the price of, and buy a given stock symbol.

Following are two SOAP message examples that the application can receive, with accompanying HTTP Request headers.

The first message sample contains a SOAP message for a GetQuote operation, requesting a quote for IBM. This is a read-only operation that gets its data from the back end, and is a good candidate for caching. In this example the SOAP message is cached and a timeout is placed on its entries to guarantee the quotes it returns are current.

Message example 1

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-lookup
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:getQuote xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:getQuote>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

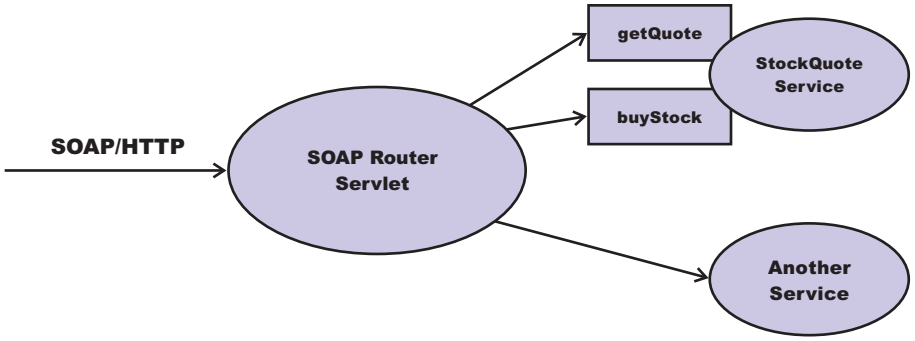
The SOAPAction HTTP header in the request is defined in the SOAP specification and is used by HTTP proxy servers to dispatch requests to particular HTTP servers. WebSphere Application Server dynamic cache can use this header in its cache policies to build IDs without having to parse the SOAP message.

Message example 2 illustrates a SOAP message for a BuyQuote operation. While message 1 is cacheable, this message is not, because it updates the back end database.

Message example 2

```
POST /soap/servlet/soaprouter
HTTP/1.1
Host: www.myhost.com
Content-Type: text/xml; charset="utf-8"
SOAPAction: urn:stockquote-update
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
<m:buyStock xmlns:m="urn:stockquote">
<symbol>IBM</symbol>
</m:buyStock>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The following graphic illustrates how to invoke methods with the SOAP messages. In Web services terms, especially Web Service Definition Language (WSDL), a service is a collection of operations such as getQuote and buyStock. A body element namespace (urn:stockquote in the example) defines a service, and the name of the first body element indicates the operation.



The following is an example of WSDL for the getQuote operation:

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
targetNamespace="http://www.getquote.com/StockQuoteService-interface"
xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns=soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
<message name="SymbolRequest">
<part name="return" type="xsd:string"/>
</message>
<portType name="StockQuoteService">
<operation name="getQuote">
<input message="tns:SymbolRequest"/>
<output message="tns:QuoteResponse"/>
</operation>
</portType>
<binding name="StockQuoteServiceBinding"
type="tns:StockQuoteService">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getQuote">
<soap:operation soapAction="urn:stockquote-lookup"/>
<input>
<soap:body use="encoded" namespace="urn:stockquote"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</input>
<output>
<soap:body use="encoded" namespace="urn:stockquotes"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
</output>
</operation>
</binding>
</definition>
```

To build a set of cache policies for a Web services application, configure WebSphere Application Server dynamic cache to recognize cacheable service operation of the operation.

WebSphere Application Server inspects the HTTP request to determine whether or not an incoming message can be cached based on the cache policies defined for an application. In this example, buyStock and stock-update are not cached, but stockquote-lookup is cached. In the `cachespec.xml` file for this Web application, the cache policies need defining for these services so that the dynamic cache can handle both SOAPAction and service operation.

WebSphere Application Server uses the operation and the message body in Web services cache IDs, each of which has a component associated with them. Therefore, each Web services `<cache-id>` rule contains only two components. The first is for the operation. Because you can perform the stockquote-lookup operation by either using a SOAPAction header or a service operation in the body, you

must define two different <cache-id> elements, one for each method. The second component is of type ″body″, and defines how WebSphere Application Server should incorporate the message body into the cache ID. You can use a hash of the body, although it is legal to use the literal incoming message in the ID.

The incoming HTTP request is analyzed by WebSphere Application Server to determine which of the <cache-id> rules match. Then, the rules are applied to form cache or invalidation IDs.

The following is sample code of a cachespec.xml file defining SOAPAction and servicesOperation rules:

```
<cache>
<cache-entry>
 <class>webservice</class>
 <name>/soap/servlet/soaprouter</name>
 <sharing-policy>not-shared</sharing-policy>
 <cache-id>
  <component id="" type="SOAPAction">
   <value>urn:stockquote-lookup</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1<priority>
  </component>
 </cache-id>
 <cache-id>
  <component id="" type="serviceOperation">
   <value>urn:stockquote:getQuote</value>
  </component>
  <component id="Hash" type="SOAPEnvelope"/>
   <timeout>3600</timeout>
   <priority>1</priority>
  </component>
 </cache-id>
</cache-entry>
</cache>
```

Dynamic cache MBean statistics

The dynamic cache service provides an MBean interface to access cache statistics.

Access cache statistics with the MBean interface, using JACL

- Obtain the MBean identifier with the **queryNames** command, for example:

  ```
  $AdminControl queryNames type=DynaCache,*  // Returns a list of the available dynamic cache MBeans
  ```

  Select your dynamic cache MBean and run the following command:

  ```
  set mbean <dynamic_cache_mbean>
  ```

- Retrieve the names of the available cache statistics:

  ```
  $AdminControl invoke $mbean getCacheStatisticNames
  ```

- Retrieve the names of the available cache instances:

  ```
  $AdminControl invoke $mbean getCacheInstanceNames
  ```

- Retrieve all of the available cache statistics for the base cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics
  ```

- Retrieve all of the available cache statistics for the named cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics "services/cache/servletInstance_4"
  ```

- Retrieve cache statistics that are specified by the names array for the base cache instance:

  ```
  $AdminControl invoke $mbean getCacheStatistics
   {"DiskCacheSizeInMB ObjectsReadFromDisk4000K RemoteObjectMisses"}
  ```

  Note: This command should all be entered on one line. It is broken here for printing purposes.

- Retrieve cache statistics that are specified by the names array for the named cache instance:

```
$AdminControl invoke $mbean getCacheStatistics
 {services/cache/servletInstance_4 "ExplicitInvalidationsLocal CacheHits"}
```

Note: This command should all be entered on one line. It is broken here for printing purposes.

- Retrieve all the cache IDs in memory for the named cache instance that matches the specified regular expression:

```
$AdminControl invoke $mbean getCacheIDsInMemory {services/cache/servletInstance_4 \S}
```

- Retrieve all cache IDs on disk for the named cache instance that matches the specified regular expression:

```
$AdminControl invoke $mbean getCacheIDsOnDisk {services/cache/servletInstance_4 \S}
```

- Retrieves the CacheEntry, which holds metadata information for the cache ID:

```
$AdminControl invoke $mbean getCacheEntry {services/cache/servletInstance_4 cache_id_1}
```

- Invalidates all cache entries that match the pattern-mapped cache IDs in the named cache instance and all cache entries dependent upon the matched entries in the instance:

```
$AdminControl invoke $mbean invalidateCacheIDs {services/cache/servletInstance_4 cache_id_1 true}
```

Example: Configuring the dynamic cache service

This example puts all of the steps together for configuring the dynamic cache service with the `cachespec.xml` file, showing the use of the cache ID generation rules, dependency IDs, and invalidation rules.

Suppose that a servlet manages a simple news site. This servlet uses the query parameter ″action″ to determine if the request views (query parameter ″view″) news or updates (query parameter ″update″) news (used by the administrator). Another query parameter ″category″ selects the news category. Suppose that this site supports an optional customized layout that is stored in the user's session using the attribute name ″layout″. Here are example URL requests to this servlet:

http://*yourhost/yourwebapp*/newscontroller?action=view&category=sports (Returns a news page for the sports category)

http://*yourhost/yourwebapp*/newscontroller?action=view&category=money (Returns a news page for the money category)

http://*yourhost/yourwebapp*/newscontroller?action=update&category=fashion (Allows the administrator to update news in the fashion category)

Here are the steps for configuring the dynamic cache service for this example with the `cachespec.xml` file:

1. Define the <cache-entry> elements that are necessary to identify the servlet. In this case, the URI for the servlet is ″newscontroller″, so this is the cache-entry <name> element. Because this example caches a servlet or JavaServer Pages (JSP) file, the cache entry class is ″servlet″.

```
<cache-entry>
<name> /newscontroller </name>
<class>servlet  </class>
 </cache-entry>
```

2. Define cache ID generation rules. This servlet caches only when action=view, so one component of the cache ID is the parameter ″action″ when the value equals ″view″. The news category is also an essential part of the cache ID. The optional session attribute for the user's layout is included in the cache ID. The cache entry is now:

```
<cache-entry>
 <name> /newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
```

```
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
</cache-entry>
```

3. Define dependency ID rules. For this servlet, a dependency ID is added for the category. Later, when the category is invalidated due to an update event, all views of that news category are invalidated. Following is an example of the cache entry after adding the dependency ID:

```
<cache-entry>
 <name>newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
 <dependency-id>category
  <component id="category" type="parameter">
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

4. Define invalidation rules. Because a category dependency ID is already defined, define an invalidation rule to invalidate the category when action=update. To incorporate the conditional logic, add ″ignore-value″ components into the invalidation rule. These components do not add to the output of the invalidation ID, but only determine whether or not the invalidation ID creates and runs. The final cache-entry now looks like the following:

```
<cache-entry>
 <name>newscontroller </name>
 <class>servlet  </class>
  <cache-id>
  <component id="action" type="parameter">
   <value>view</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
   <required>true</required>
  </component>
  <component id="layout" type="session">
   <required>false</required>
  </component>
 </cache-id>
 <dependency-id>category
  <component id="category" type="parameter">
   <required>true</required>
  </component>
 </dependency-id>
 <invalidation>category
  <component id="action" type="parameter" ignore-value="true">
   <value>update</value>
   <required>true</required>
  </component>
  <component id="category" type="parameter">
```

```
      <required>true</required>
        </component>
     </invalidation>
    </cache-entry>
```

Dynamic cache PMI counter definitions

The dynamic cache statistics interface is defined as WSDynamicCacheStats under the `com\ibm\websphere\pmi\stat` package.

Dynamic cache statistics are structured as follows in the Performance Monitoring Infrastructure (PMI) tree:

```
__Dynamic Caching+
  |
  |__<Servlet: instance_1>
    |__Templates+
      |__<template_1>
      |__<template_2>
    |__Disk+
      |__<Disk Offload Enabled>
  |
  |__<Object: instance_2>
    |__Object Cache+
      |__<Counters>
+ indicates logical group
```

StatDescriptor locates and accesses particular statistics in the PMI tree. For example:

1. StatDescriptor to represent statistics for cache servlet: `instance_1 templates group template_1: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Servlet: instance1", WSDynamicCacheStats.TEMPLATE_GROUP, "template_1"});`
2. StatDescriptor to represent statistics for cache servlet: `instance_1 disk group Disk Offload Enabled: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Servlet: instance_1", WSDynamicCacheStats.DISK_GROUP, WSDynamicCacheStats.DISK_OFFLOAD_ENABLED});`
3. StatDescriptor to represent statistics for cache object: `instance2 object cache group Counters: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Object: instance_2", WSDynamicCacheStats.OBJECT_GROUP, WSDynamicCacheStats.OBJECT_COUNTERS});`

Note: Cache instance names are prepended with cache type ("Servlet: " or "Object: ").

Counter definitions for Servlet Cache

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED`	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED`	The number of requests for cacheable objects that are served from disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1
WSDynamicCacheStats. DependencyIdsBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1 " - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

Counter definitions for Object Cache

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of requests for cacheable objects that are served from disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1
WSDynamicCacheStats. DependencyIds BufferedForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats.DISK_ OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP / -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBasedInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

Using the dynamic cache service

Use the dynamic cache service to improve application performance by caching the output of servlets, Web services, and WebSphere Application Server commands into memory.

Before you begin

Develop a cache policy for your application. The cache policy defines rules for what responses to cache and the amount of time the responses should be held in the cache. See “Configuring cacheable objects with the cachespec.xml file” on page 2231 for more information.

About this task

The dynamic cache service is enabled by default. You can configure the default cache instance, as follows:

1. Click **Servers > Server Types > WebSphere® application servers > server_name > Container services > Dynamic cache service**.
2. Configure the default cache instance or follow the links to enable servlet or portlet caching. See “Dynamic cache service settings” on page 2211 for more information about default cache settings.

What to do next

You might want to enable dynamic cache disk offload. This option moves cache entries that are expired from memory to disk for potential future access. See “Configuring dynamic cache disk offload” on page 2222 for more information.

Dynamic cache service settings

Use this page to configure and manage the dynamic cache service settings.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Container services > Dynamic cache service**.

Enable servlet caching:

The dynamic servlet cache service starts when servlet caching is enabled in Web Container panel.

Enable portlet caching:

Start the dynamic portlet cache service by enabling servlet caching, then, enabling portlet fragment caching under Portlet Container panel.

Cache provider:

Specifies whether to configure the server to use the dynamic cache or a stack product cache provider.

Cache size:

Specifies a positive integer as the value for the maximum number of entries that the cache holds.

Enter a cache size value in this field that is between the range of 100 through 200,000.

Default priority:

Specifies the default priority for cache entries, determining how long an entry stays in a full cache.

Default	1
Range	1 to 255

Limit memory cache size:

Specifies the size of the memory cache.

Use this feature to provide an ability to constrain the cache in terms of the JVM heap. In addition to specifying the cache size in MB, dynamic cache will also enable you to set a high watermark and low watermark for the cache heap that is consumed. When the cache heap memory reaches the high watermark, dynamic cache will either discard or LRU to disk, until the cache is brought down to the low watermark. This functionality of limiting the cache in terms of the JVM heap is only available if the objects that are put into the cache implement the sizeable interface. This interface has one method that returns the size of the object in bytes put into the cache. Dynamic cache will use the sizeable interface to estimate the heap size of the cache.

Default	-1 to disable limiting the memory cache size
Range	1 to maximum integer

Memory cache size:

Specifies a value for the maximum memory cache size in megabytes (MB).

High threshold:

Specifies a high watermark when the memory cache eviction policy starts. The threshold is expressed in terms of the percentage of the memory cache size in megabytes (MB). The default value is 95%

Values	1 to 100

Low threshold:

Specifies a low watermark when the memory cache eviction policy ends. The threshold is expressed in terms of the percentage of the memory cache size in megabytes (MB). The default value is 80%.

Values	1 to 100

Enable disk offload:

Specifies whether disk offload is enabled.

By default, the dynamic cache maintains the number of entries that are configured in memory. If new entries are created while the cache is full, the priorities that are configured for each cache entry, and a least recently used algorithm, are used to remove entries from the cache. In addition to having a cache entry removed from memory when the cache is full, you can enable disk offload to have a cache entry copied to the file system (the location is configurable). Later, if that cache entry is needed, it is moved back to memory from the file system.

Before you enable disk offload, consider the following:

- You cannot specify the number of cache entries that are offloaded to disk.
- You cannot specify the amount of disk space to use.

Offload location:

Specifies the location on the disk to save cache entries when disk offload is enabled.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/node/server name/_dynacache/cache JNDI name` is used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/node/server name/cache JNDI name`. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The application server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all of the disk cache content at the old location is lost when you restart the application server

When you are specifying a directory, consider the following:

- If you use the default directory and the disk fills up, WebSphere Application Server could possibly stall if it needs to write messages to log files, and there is no more space.
- Depending on the operating system, you may see disk full messages on the console.

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if **Enable disk offload** is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	3 and above.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Disk cache performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom performance	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is CUSTOM.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	10 to MAXINT

Disk cache eviction algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache. This setting does not apply when the disk cache eviction policy is set to none.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value is used when limit disk cache size in GB and limit disk cache size in entries are specified. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy will end. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value is used limit disk cache size in GB and limit disk cache size in entries are specified. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Configuring servlet caching

After a servlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the servlet. These side effects can include calls to other servlets or JavaServer Pages (JSP) files or metadata about the entry, including timeout and entry priority information. Configure servlet caching to save the output of servlets and JavaServer Pages (JSP) files to the dynamic cache.

Before you begin

To enable servlet caching, you must complete “Using the dynamic cache service” on page 2210.

About this task

Unique entries are distinguished by an ID string that is generated from the HttpServletRequest object each time the servlet runs. You can then base servlet caching on:

- Request parameters and attributes of the Universal Resource Identifier (URI) that was used to invoke the servlet
- Session information
- Other options, including cookies

Because JavaServer Pages files are compiled into servlets, the dynamic cache function treats JavaServer Pages files the same as servlets, except in specifically documented situations.

1. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Web container settings > Web container** in the console navigation tree.
2. Select **Enable servlet caching** under the Configuration tab.
3. Click **Apply** or **OK**.
4. Restart WebSphere Application Server. See Managing application servers for more information.

What to do next

Define the cache policy for your servlets by “Configuring cacheable objects with the cachespec.xml file” on page 2231.

Dynamic caching with Asynchronous Request Dispatcher:

Asynchronous Request Dispatcher (ARD) improves servlet response time when slow operations are logically separated and performed concurrently with other operations that are required to complete the response.

Servlet caching works with ARD include requests, with certain caveats. The dynamic caching service does not support the ESI and ExternalCache features when ARD is enabled, due to complex and intractable buffering issues with third party content.

If the include request has been cached by the dynamic cache service, the include request returns immediately with the response data written to the top-level response data for the servlet.

In combination with the Remote Request Dispatcher (RRD), include request processing can also be offloaded to other members in the application server core group, thus reducing resource requirements on the original application server.

Configuring portlet fragment caching

After a portlet is invoked and completes generating the output to cache, a cache entry is created, containing the output and the side effects of the portlet. These side effects can include calls to other portlets or metadata about the entry, including timeout and entry priority information. Configure portlet fragment caching with the WebSphere Application Sever administrative console to save the output of portlets to the dynamic cache.

Before you begin

To enable portlet fragment caching, you must complete "Using the dynamic cache service" on page 2210.

About this task

Unique entries are distinguished by an ID string that generates from the PortletRequest object each time the portlet runs. You can then base portlet fragment caching on:

- Request parameters and attributes
- Session information
- Portlet-specific information, portlet session, portlet window ID, portlet mode, and portlet window state

1. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Portlet container settings > Portlet container** in the administrative console navigation tree.
2. Select **Enable portlet fragment cache** under the Configuration tab.
3. Click **Apply** or **OK**.
4. Restart WebSphere Application Server.

   See the *Setting up the application serving environment* PDF for more information.

What to do next

Define a cache policy for your portlets. Note that portlets are not cached unless an applicable caching policy is defined in a cachespec.xml file. See "Configuring cacheable objects with the cachespec.xml file" on page 2231 for general task information about defining a cache policy. See "Configuring caching policies for portlets" for information about defining portlet-specific aspects in a cache policy.

Configuring caching policies for portlets:

Fragment caching for portlets requires that you define a cache policy in a cachespec.xml file, either within the portlet Web application archives (WAR) file or globally. If no caching policy is defined and applicable to a particular portlet, that porlet is not cached.

WebSphere Application Server caching policies provide a lot of flexibility for defining cache IDs and invalidation rules that match the specific requirements of individual portlets. The caching policies that you can define are not necessarily compliant with the caching behavior that is defined by the Java Portlet Specification. The following sections provide some recommendations on how you can exploit the features of cachespec.xml file, to define a caching policy that conforms to the specification.

Cache expiration. Portlets define cache expiration time in the `<expiration-cache>` element of the portlet.xml deployment descriptor. If this element is not present, or has a value of zero, the portlet is not cached. The cache expiration time for portlets is only defined in the deployment descriptor; any cache timeout values that are specified in a cachespec.xml file have no effect.

Caching scopes. Portlets are defined in the <caching-scope> element of the portlet.xml deployment descriptor, whether the portlet content should be shared across all users or whether it contains user-specific information and must be cached individually for each user. To maintain this setting in your caching policy definition, include the com.ibm.wsspi.portletcontainer.user_cache_scope attribute in your cache key, with the following cache key component:

```
<component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>
```

This attribute has the following values:

- The value, public, in a portlet that defines public cache scope.
- The current logon user ID in a portlet that defines private cache scope.
- Null (anonymous) in a portlet that defines private cache scope if no user is logged on.

If you want to cache portlet content for anonymous access, even in a portlet that defines private cache scope, add <required>false</required> to the cache key component. This implies that all anonymous browser access will retrieve the same cache content.

Portlet lifecycle methods. The Java Portlet Specification defines the four lifecycle phases: action, event, render and resource for running in a portlet. Only the render and resource phases produce content; the action and event phases invoke portlet activity without generating content and must not be cached. The lifecycle phase for a portlet call is available in the javax.portlet.lifecycle_phase request attribute. Check for the correct lifecycle by including the following cache key component:

```
<component id="javax.portlet.lifecycle_phase" type="attribute">
 <value>RENDER_PHASE</value>
</component>
```

This cache key component only caches render requests to the portlet. Cache additional resource requests by adding the RESOURCE_PHASE. In many cases, the best approach is to define a separate <cache-id> element for resource requests. The resource ID is available in the com.ibm.wsspi.portletcontainer.resource_id request attribute for caching key generation in resource requests.

Request parameters. Portlets can typically display multiple views. Render parameters distinguish which view displays. Each combination of parameters addresses a different view of the portlet. All views need to be cached separately; therefore, the full request parameter map should normally be included in the cache key. The com.ibm.wsspi.portletcontainer.all_parameters attribute provides a unique value for the content of the full request parameter map that can be used with the following cache key component:

```
<component id="com.ibm.wsspi.portletcontainer.all_parameters" type="attribute">
 <required>false</required>
</component>
```

If you write a cache policy for a specific portlet, and you know exactly which views of the portlet are addressed by which request parameters, it is usually more efficient to use specific <parameter> elements in the cache key to cache only the most important views of the portlet.

Other cache key components. Depending on your usage scenario, you will need to include other information in your cache key, if the returned content depends on it (for example, the portlet mode and window state, or the request locale in a multi-language portal). In a multi-device portal that supports different markup types, the returned content type should also be part of the cache key. The content type for a portlet is available in the com.ibm.wsspi.portletcontainer.response_contenttype request attribute.

Cache invalidation. The Java Portlet Specification states that action and event requests to a portlet must invalidate all currently cached content. The portlet caching definitions usually allow for caching multiple views of a portlet at the same time. To invalidate all of them, use the dependency ID mechanism of cachespec.xml.

Define a common dependency ID for all views that should be invalidated by an action. The common ID will usually only include the portlet window ID and the user scope, so that a portlet action does not affect private cache entries for other users:

```
<dependency-id>action
 <component id="" type="portletWindowId"/>
 <component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>
</dependency-id>
```

Define an invalidation rule that repeats the dependency ID and adds the current lifecycle method as a condition. It is essential to have the `ignore-value` attribute on the condition part. The lifecycle attribute must not be part of the returned invalidation ID, because that invalidation ID must match exactly with the dependency ID that is specified above.

```
<invalidation>action
 <component id="" type="portletWindowId"/>
 <component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>
 <component id="javax.portlet.lifecycle_phase" type="attribute" ignore-value="true">
  <value>ACTION_PHASE</value>
  <value>EVENT_PHASE</value>
 </component>
</invalidation>
```

Following the same pattern, specify more complex invalidation rules in caching policies for individual portlets, (for example, you can only invalidate a subset of the portlet views for specific actions that are determined by request parameters). The following example code describes a generic caching configuration that conforms to the behavior that is defined by the Java Portlet Specification:

Sample cachespec.xml file

```
<?xml version="1.0" ?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
 <cache-entry>
  <class>portlet</class>
  <name>MyPortlet</name>
  <property name="consume-subfragments">true</property>
  <cache-id>
   <component id="" type="portletWindowId"/>
   <component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>

   <component id="" type="portletWindowState">
   <!-- minimized portlets are not cached -->
   <not-value>minimized</not-value>
  </component>
  <component id="" type="portletMode"/>

  <component id="" type="locale"/>
  <component id="com.ibm.wsspi.portletcontainer.response_contenttype" type="attribute"/>

  <component id="com.ibm.wsspi.portletcontainer.all_parameters" type="attribute">
   <required>false</required>
  </component>

  <component id="javax.portlet.lifecycle_phase" type="attribute">
   <value>RENDER_PHASE</value>
  </component>
 </cache-id>

 <dependency-id>action
  <component id="" type="portletWindowId"/>
  <component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>
 </dependency-id>

 <invalidation>action
  <component id="" type="portletWindowId"/>
```

```
  <component id="com.ibm.wsspi.portletcontainer.user_cache_scope" type="attribute"/>
  <component id="javax.portlet.lifecycle_phase" type="attribute" ignore-value="true">
   <value>ACTION_PHASE</value>
   <value>EVENT_PHASE </value>
  </component>
 </invalidation>

 </cache-entry>
</cache>
```

Configuring portlet fragment caching with the wsadmin tool

You can configure portlet fragment caching with scripting and the wsadmin tool.

Before you begin

Before starting this task, the wsadmin tool must be running. See the *Using the administrative clients* PDF for more information.

About this task

Note: If you use the wsadmin tool to enable portlet fragment caching, you must make sure that servlet caching is also enabled. Similarly if you use the wsadmin tool to disable portlet fragment caching, you must make sure that servlet caching is also disabled. The settings for these two caching functions must stay synchronized. If you enable or disable portlet fragment caching using the administrative console, synchronization is automatically taken care of for you.

1. Locate the server object. The following example selects the first server found:

   Using Jacl:

   ```
   set s1 [$AdminConfig getid /Server:server1/]
   ```

   Using Jython:

   ```
   s1 = AdminConfig.getid('/Server:server1/')
   ```

2. List the Web containers and assign them to the `wc` variable, for example:

   Using Jacl:

   ```
   set wc [$AdminConfig list PortletContainer $s1]
   ```

   Using Jython:

   ```
   wc = AdminConfig.list('PortletContainer', s1)
   ```

3. Set the enablePortletCaching attribute to `true` and assign it to the `serEnable` variable, for example:

   Using Jacl:

   ```
   set serEnable "{enablePortletCaching true}"
   ```

   Using Jython:

   ```
   serEnable = [['enablePortletCaching', 'true']]
   ```

4. Enable caching, for example:

   Using Jacl:

   ```
   $AdminConfig modify $wc $serEnable
   ```

   Using Jython:

   ```
   AdminConfig.modify(wc, serEnable)
   ```

Configuring caching for Struts and Tiles applications

Use this task to cache Struts and Tiles applications.

Before you begin

Before you configure Struts and Tiles caching, you should have a developed application. For more information about developing Struts and Tiles applications, see The Apache Struts Web Application

Framework on the Apache Web site at http://struts.apache.org/.

About this task

Use this task when you want to cache data in Struts and Tiles applications.

Struts is an open source framework for building Web applications using the Model-View-Controller (MVC) architecture. The Struts framework has a controller component and integrates with other technologies to provide the model and the view. Struts provide a control layer for the Web application, which reduces construction time and maintenance costs.

The Tiles framework builds on the jsp:include feature and is bundled with the Struts Web application framework. The Tiles framework reduces the duplication between JavaServer Pages (JSP) files and makes Web site layouts flexible and easy to maintain by assembling presentation pages from component parts.

Struts and Tiles caching is an extension of servlet and JSP caching, so the actions performed for each type of caching are very similar. See "Configuring servlet caching" on page 2215 for more information.

1. Enable servlet and JSP caching. Enabling servlet caching automatically enables Struts and Tiles caching. See "Configuring servlet caching" on page 2215 for more information.
2. Develop the cache policy. A cache policy is required to cache a struts or tiles response.

   To develop a Struts cache policy:

   The Struts framework provides the controller component in the MVC-style application. The controller is a servlet called `org.apache.struts.action.ActionServlet.class`. In the `web.xml` file of the application, a servlet mapping of `*.do` is added for this Struts ActionServlet servlet so that every request for a Web address that ends with `.do` is processed. The ActionServlet servlet uses the information in the `struts-config.xml` file to decide which Struts action class runs the request for the specified resource.

   Cache policy using a previous version of WebSphere Application Server

   In the previous version of WebSphere Application Server, only one cache policy per servlet was supported. However, when you are using Struts, every request that ends in `.do` maps to the same ActionServlet servlet. To cache Struts responses, write a cache policy for the ActionServlet servlet based on its servlet path.

   For example, consider two Struts actions: `/HelloParam.do` and `/HelloAttr.do`. To cache the responses based on the id request parameter and the arg request attribute respectively, use the following cache policy:

```
<cache-entry>
 <class>servlet</class>
 <name>org.apache.struts.action.ActionServlet.class</name>
 <cache-id>
  <component id="" type="servletpath">
   <value>/HelloParm.do</value>
  </component>
 </cache-id>
 <cache-id>
  <component id="" type="servletpath">
   <value>/HelloAttr.do</value>
  </component>
  <component id="arg" type="attribute">
   <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

   Cache policy using WebSphere Application Server, Version 6.0 or later

With the current version of WebSphere Application Server, you can map multiple cache policies for a single servlet. You can rewrite the previous cache policy as in the following example:

```
<cache-entry>
 <class>servlet>
 <name>/HelloParam.do</name>
 <cache-id>
  <component id="id" type="parameter">
   <required>true</required>
  </component>
</cache-entry>
<cache-entry>
 <class>servlet</class>
 <name>/HelloAttr.do</name>
 <cache-id>
  <component id="arg" type="attribute">
   <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

To develop a Tiles cache policy:

The Tiles framework is built on the jsp:include tag, so everything that applies to JSP caching also applies to Tiles. You must set the flush attribute to `true` in any fragments that are included using the tiles:insert tag for the fragments to be cached correctly. The extra feature in tiles caching over JSP caching is based on the tiles attribute. For example, you might develop the following layout.jsp template:

```
<html>
 <%String categoryId = request.getParameter("categoryId")+"test"; %>
 <tiles:insert attribute="header">
  <tiles:put name="categoryId" value="<%= categoryId %>" />
 </tile:insert>
 <table>
  <tr>
   <td width="70%" valign="top"><tiles:insert attribute="body" /> </td>
  </tr>
  <tr>
   <td colspan="2"><tiles:insert attribute="footer" /></td>
  </tr>
 </table>
</body>
</html>
```

The nested tiles:put tag specifies the attribute of the inserted tile. In the `layout.jsp` template, the categoryId attribute is defined and passed on to the tile that is inserted into the placeholder for the header. In the following example, the `layout.jsp` file is inserted into another JSP file:

```
<html>
<body>
<tiles:insert page="layout.jsp?categoryId=1002" flush="true">
 <tiles:put name="header" value="/header.jsp" />
 <tiles:put name="body" value="/body.jsp" />
 <tiles:put name="footer" value="/footer.jsp" />
</tiles:insert>
</body>
</html>
```

The categoryId tile attribute is passed on to the `header.jsp` file. The `header.jsp` file can use the <tiles:useAttribute> tag to retrieve the value of categoryId. To cache the `header.jsp` file based on the value of the categoryId attribute, you can use the following cache policy:

```
<cache-entry>
 <class>servlet</class>
 <name>/header.jsp</name>
 <cache-id>
```

```
  <component id="categoryId" type="tiles_attribute">
  <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

3. Ensure your cache policy is working correctly. You can modify the policies within the `cachespec.xml` file while your application is running. See “Configuring cacheable objects with the cachespec.xml file” on page 2231 for more information about cache policies.

Results

What to do next

See “Task overview: Using the dynamic cache service to improve performance” on page 2197 for more information about the dynamic cache.

Related tasks

“Configuring servlet caching” on page 2215
After a servlet is invoked and completes generating the output to cache, a cache entry is created containing the output and the side effects of the servlet. These side effects can include calls to other servlets or JavaServer Pages (JSP) files or metadata about the entry, including timeout and entry priority information. Configure servlet caching to save the output of servlets and JavaServer Pages (JSP) files to the dynamic cache.

“Configuring cacheable objects with the cachespec.xml file” on page 2231
Use this task to define cacheable objects inside the `cachespec.xml`, found inside the Web module `WEB-INF` or enterprise bean `META-INF` directory.

“Task overview: Using the dynamic cache service to improve performance” on page 2197
Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance. WebSphere Application Server consolidates several caching activities including servlets, Web services, and WebSphere commands into one service called the *dynamic cache*. These caching activities work together to improve application performance, and share many configuration parameters that are set in the dynamic cache service of an application server. You can use the dynamic cache to improve the performance of servlet and JSP files by serving requests from an in-memory cache. Cache entries contain servlet output, the results of a servlet after it runs, and metadata.

Configuring dynamic cache disk offload

Use this task to configure dynamic cache disk offload, which saves cache entries that are deleted from the memory cache to disk.

About this task

By default, when the number of cache entries reaches the configured limit for a given application server, cache entries are removed from the memory cache, allowing newer entries to be stored in the cache. Use disk offload to copy the cache entries that are being removed from the memory cache to disk for potential future access.

1. In the administrative console, click **Servers** > **Server Types** > **WebSphere application servers >** *server_name* **> Container services > Dynamic cache service**.
2. Select **Enable disk offload**.
3. After you enable the disk offload, you can set the **Disk offload location**. The disk offload location specifies where to save the cache entries on the disk. The disk offload location must be unique for any application servers that are defined on the same node. If you have multiple servers defined on the same node, make sure the disk offload location is different for each server.
4. Enable **Flush to disk** if you want cache objects that are in memory to be saved to disk when the server is stopped. Disk offload must be enabled if you choose this option. If you do not enable flush to disk, all the cache objects are deleted when the server stops.

5. Click **Apply** or **OK**.
6. Restart WebSphere Application Server.

Results

You enabled disk offload. Memory cache entries are moved to disk for potential future access.

When you have two or more application servers with servlet caching enabled and the application servers specify the same disk offload location for their caches through the dynamic cache service, the following exceptions might occur:

```
java.lang.NullPointerException
       at com.ibm.ws.cache.CacheOnDisk.readTemplate(CacheOnDisk.java:686)
       at com.ibm.ws.cache.Cache.internalInvalidateByTemplate(Cache.java:828)
```

or:

```
java.lang.NullPointerException
       at com.ibm.ws.cache.CacheOnDisk.readCacheEntry(CacheOnDisk.java:600)
       at com.ibm.ws.cache.Cache.getCacheEntry(Cache.java:341)
```

If one server is run as root and the other servers are run as non-root, this problem could occur. For example, if `server1` runs as root and `server2` runs as `wasuser` or `wasgroup`, the cache files in the disk offload location might be created with root permissions. This situation causes the applications running on the non-root servers to crash when they try to read or write to the cache.

Java virtual machine cache settings:

Use this page to set Java virtual machine (JVM) custom properties to maintain cache entries that are saved to disk.

You can set the custom properties globally to affect all cache instances, or you can set the custom property on a single cache instance. In most cases, set the properties on the individual cache instances. To set the custom properties on the default cache instance, use the global option. If you set the same property both globally and on a cache instance, the value that is set on the cache instance overrides the global value.

To configure the custom properties on a single object cache instance or servlet cache instance, perform the following steps:

1. In the administrative console, click one of the following paths:
   - To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name* **> Custom properties > New**.
   - To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name* **> Custom properties > New**.
2. Type the name of the custom property. When configuring these custom properties on a single cache instance, you do not use the full property path. For example, type `explictBufferLimitOnStop` to configure the `com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop` custom property.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

To configure the custom property globally across all configured cache instances, perform the following steps:

1. In the administrative console, click **Servers > Application servers >** *server_name* **> Java and process management > Process definition > Java virtual machine > Custom properties > New**.
2. Type the name of the custom property (`com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop`) in the **Name** field.

3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

Also use these properties to tune the delay offload function for the disk cache.

Note: Setting these custom properties using the **wsadmin** command is deprecated for WebSphere Application Server Version 7.0. Use the administrative console to set these properties. The individual property descriptions include information on how to use the administrative console to set these properties.

The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory. Use any combination of the following properties to tune your configuration:

- To increase or decrease the in-memory limit of cache IDs for dependency ID and template buffers, use the `com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit` custom property.
- To disable the disk cache delay offload function, use the `com.ibm.ws.cache.CacheConfig.htodDelayOffload` custom property. Disabling this property saves all cache entries to disk immediately after removing them from the memory cache.

com.ibm.ws.cache.CacheConfig.htodCleanupFrequency:

Use this property to change the amount of time between disk cache cleanup.

Note: Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field if it is not already selected.
2. Under Performance Settings, select Balanced performance and balanced memory usage or Custom.
3. In the Disk cache cleanup frequency field, specify an appropriate length of time, in minutes.

By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable. Use the `com.ibm.ws.cache.CacheConfig.htodCleanupFrequency` custom property to change the time interval between disk cache cleanup.

Units	minutes For example, a value of 60 means 60 minutes between each disk cache cleanup.
Default	0 The disk cache cleanup occurs at midnight every 24 hours.

com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit:

Use this property to specify the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache.

Note: Setting this custom property using the **wsadmin** command is deprecated for V7.0. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field, if it is not already selected.
2. Under Disk Cache settings, select Limit disk cache size in entries, if it is not already selected.
3. In the Disk cache size field, specify the number of cache IDs that can be saved in memory for the dependency ID and template buffers.

Units	number of cache IDs For example, a value of 1000 means that each dependency ID or template ID can have up to 1000 different cache IDs in memory.
Default	1000
Minimum	100

com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop:

Use this custom property when the flush-to-disk-on-stop feature is enabled. When the server is stopping, offloads are limited to the value specified for this property, pending removal of entries in the explicit invalidation buffer.

If this property is set to `0`, there is no limit to the number of offloads that can occur. Only positive integers are accepted as values for this property. If the number of entries in the explicit invalidation buffer is greater than the specified limit, all of the disk files for this specified cache instance are deleted after the server stops.

Note: You cannot use the administrative console to set this property.

com.ibm.ws.cache.CacheConfig.lruToDiskTriggerTime:

Use this custom property to set the frequency with which cache entries in memory are asynchronously offloaded to disk when the disk offload feature is enabled.

Units	integer, milliseconds
Lower bound	0
Upper bound	5000
Scope	Applicable to all cache instances.

com.ibm.ws.cache.CacheConfig.lruToDiskTriggerPercent:

Use this custom property to set the percentage of the memory cache size to be used as an overflow buffer when disk offload is enabled.

Cache entries in the overflow buffer are purged and asynchronously offloaded to disk at a frequency of *lruToDiskTriggerTime* milliseconds. If the memory overflow buffer is full, cache entries are offloaded to disk synchronously on the thread for the caller.

Units	integer, percentage
Lower bound	0
Upper bound	100
Scope	Configurable per cache instance.

Configuring Edge Side Include caching

The Web server plug-in contains a built-in ESI processor. The ESI processor can cache whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache, therefore, the cache entries are not saved when the Web server is restarted.

About this task

Edge Side Include (ESI) is configured through the `plugin-cfg.xml` file.

When a request is received by the Web server plug-in, it is sent to the ESI processor, unless the ESI processor is disabled. It is enabled by default. If a cache miss occurs, a Surrogate-Capabilities header is added to the request and the request is forwarded to the WebSphere Application Server. If servlet caching is enabled in the application server, and the response is edge cacheable, the application server returns a Surrogate-Control header in response to the WebSphere Application Server plug-in.

The value of the Surrogate-Control response header contains the list of rules that are used by the ESI processor to generate the cache ID. The response is then stored in the ESI cache, using the cache ID as the key. For each ESI include tag in the body of the response, a new request is processed so that each nested include results in either a cache hit or another request that forwards to the application server. When all nested includes have been processed, the page is assembled and returned to the client.

The ESI processor is configurable through the WebSphere Web server plug-in configuration file `plugin-cfg.xml`. The following is an example of the beginning of this file, which illustrates the ESI configuration options.

```
<?xml version-"1.0"?>
<Config>
  <Property Name="esiEnable" Value="true"/>
  <Property Name="esiMaxCacheSize" Value="1024"/>
  <Property Name="esiInvalidationMonitor" Value="false"/>
```

- The first option, esiEnable, can be used to disable the ESI processor by setting the value to false. ESI is enabled by default. If ESI is disabled, then the other ESI options are ignored.
- The second option, esiMaxCacheSize, is the maximum size of the cache in 1K byte units. The default maximum size of the cache is 1 megabyte. If the cache is full, the first entry to be evicted from the cache is the entry that is closest to expiration.
- The third option, esiInvalidationMonitor, specifies if the ESI processor should receive invalidations from the application server. ESI works well when the Web servers following a threading model are used, and only one process is started. When multiple processes are started, each process caches the responses independently and the cache is not shared. This could lead to a situation where, the system's memory is fully used up by ESI processor. There are three methods by which entries are removed from the ESI cache: first, an entry expiration timeout occurs; second, an entry is purged to make room for newer entries; or third, the application server sends an explicit invalidation for a group of entries. For the third

mechanism to be enabled, the esiInvalidationMonitor property must be set to true and the DynaCacheEsi application must be installed on the application server. The DynaCacheEsi application is located in the installableApps directory and is named DynaCacheEsi.ear. If the ESIInvalidationMonitor property is set to true but the DynaCacheEsi application is not installed, then errors occur in the Web server plug-in and the request fails.

- On distributed platforms, the cache for the ESI processor is monitored through the CacheMonitor application. In order for ESI processor cache to be visible in the CacheMonitor, the DynaCacheEsi application must be installed as described above, and the ESIInvalidationMonitor property must be set to true in the `plugin-cfg.xml` file.
- When WebSphere Application Server is used to serve static data, such as images and HTML on the application server, the URLs are also cached in the ESI processor. This data has a default timeout of 300 seconds. You can change the timeout value by adding the property com.ibm.servlet.file.esi.timeOut to the Java virtual machine (JVM) command line parameters. The following example shows how to set a one minute timeout on static data cached in the plug-in:

  ```
  -Dcom.ibm.servlet.file.esi.timeOut=60
  ```

  For information about configuring alternate URL, see the *Tuning guide* PDF.

Configuring alternate URL:

Alternate URL is a method for edge caching JavaServer Pages (JSP) files and servlet responses that you can not request externally. Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge fragments that can be cached. However, you must be able to externally request an edge fragment from the application server before it can be cached. In other words, if a user types the URL in their browser with the appropriate parameters and cookies for the fragment, WebSphere Application Server must be able to return the content for that fragment.

About this task

One of the standard Java 2 Platform, Enterprise Edition (J2EE) programming architectures is the model-view-controller (MVC) architecture, where a call to a controller servlet might include one or more child JSP files to construct the view. When using the MVC programming model, the child JSP files are edge cached only if you can request these JSP files externally, which is not usually the case. For example, if a child JSP file uses one or more request attributes that are determined and set by the controller servlet, you cannot cache that JSP file on the edge. You can use alternate URL support to overcome this limitation by providing an alternate controller servlet URL used to invoke the JSP file.

The alternate URL for a JSP file or a servlet is set in the `cachespec.xml` file as a property with the name `alternate_url`. You can set the alternate URL either on a per cache-entry basis or on a per cache-id basis. It is valid only if the `EdgeCacheable` property is also set for that entry. If the `EdgeCacheable` property is not set, the `alternate_url` property is ignored. The following is a sample cache policy using the `alternate_url` property:

```
<cache-entry>
    <class>servlet</class>
    <name>/AltUrlTest2.jsp</name>
    <property  name="EdgeCacheable">true</property>
    <property  name="alternate_url">/alturlcontroller2</property>
        <cache-id>
            <timeout>600</timeout>
            <priority>2</priority>
        </cache-id>
</cache-entry>
```

What to do next

For more information on the `cachespec.xml` file, see “cachespec.xml file” on page 2233.

Configuring external cache groups

The dynamic cache can control caches outside of the application server, such as the Edge server, an IBM HTTP Server, or an HTTP Server ESI Fragment Processor plug-in.

About this task

When external cache groups are defined, the dynamic cache matches externally cacheable cache entries with those groups, and pushes cache entries and invalidations out to those groups. This allows WebSphere Application Server to manage dynamic content beyond the application server. The content can then be served from the external cache, instead of the application server, improving savings in performance.

1. Open the administrative console.
2. Enable the dynamic cache.
   a. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service**.
   b. Select **Enable service at server startup** to enable the dynamic cache each time the application server starts.
3. Define the external cache group that WebSphere Application Server should control.
   a. In the administrative console, click **Servers** > **Application servers >** *server_name* **> Container services > Dynamic cache service > External cache groups**.
   b. Click **New** or choose an external cache group from the list.
4. Configure cache group members.
   a. Click **External cache groups** from the dynamic cache administrative console page. Then click **New** or choose an external cache group from the list.
   b. Click **External cache group members** > **New** or choose an external cache group member from the list.
   c. Type the configuration string in the **Address** field.
   d. Type the adapter bean name in the **Adapter Bean Name** field.
   e. **Save** the configuration.
   f. Click **Apply** or **OK**.

External cache group collection:

Use this page to define sets of external caches that are controlled by WebSphere Application Server on Web servers such as IBM Edge Server and IBM HTTP Server.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups**.

Name:

Specifies the external cache group name.

The external cache group name needs to match the **ExternalCache** property as defined in the servlet or Java Server Pages (JSP) file `cachespec.xml` file.

When external caching is enabled, the cache matches pages with its Universal Resource Identifiers (URI) and pushes matching pages to the external cache. The entries can then be served from the external cache, instead of from the application server.

Type:

Specifies the external cache group type.

External cache group settings:

Use this page to configure sets of external caches that are controlled by WebSphere Application Server on Web servers, such as IBM Edge Server and IBM HTTP Server.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups** > *external_cache_group*.

Name:

Specifies the external cache group name.

The external cache group name must match the **ExternalCache** property as defined in the servlet or JavaServer Pages (JSPs) `cachespec.xml` file.

When external caching is enabled, the cache matches pages with its Universal Resource Identifiers (URIs) and pushes matching pages to the external cache. The entries can then be served from the external cache, instead of the application server. This ability creates a significant savings in performance.

External cache group member collection:

Use this page to define specific caches that are members of a cache group.

To view this administrative console page, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* > **Container services > Dynamic cache service** > **External cache groups** > *external_cache_group* > **External cache group members**.

Address:

Specifies a configuration string that is used by the external cache adapter bean to connect to the external cache.

AdapterBeanName:

Specifies the adapter bean name.

Example adapter bean names that are supported in WebSphere Application Server are as follows:

AFPA
AdapterBeanName: com.ibm.ws.cache.servlet.Afpa
Address: Port on which afpa listens
ESI
AdapterBeanName: com.ibm.websphere.servlet.cache.ESIInvalidatorServlet
IBM Web Traffic Express (WTE) (IBM Edge Server)
AdapterBeanName: com.ibm.websphere.edge.dynacache.WteAdapter
Address: hostname:port (host name and port on which WTE is listening)

External cache group member settings:

Use this page to define a single cache that is controlled by WebSphere Application Server.

To view this administrative console page, click **Servers > Server Types > WebSphere application servers > *server_name* > Container services > Dynamic cache service > External cache groups > *external_cache_group* > External cache group members > *external_cache_group_member***.

Advanced Fast Path Architecture adapter bean name:

Specifies the adapter bean name.

- **Adapter bean name**: specifies the adapter bean name. For example, you can use a bean name such as `com.ibm.ws.cache.servlet.Afpa`.
- **Address:** specifies the port on which AFPA listens.

Edge Side Include (ESI):

Specifies the adapter bean name.

- **Adapter bean name**: specifies the adapter bean name. For example, you can use a bean name such as `com.ibm.websphere.servlet.cache.ESIInvalidatorServlet`.
- **Address:** local host

IBM Web Traffic Express (IBM WebSphere Edge Server):

Specifies the adapter bean name.

- **Adapter bean name**: specifies the adapter bean name. For example, you can use a bean name such as `com.ibm.websphere.edge.dynacache.WteAdapter`.
- **Address:** hostname:port (host name and port on which WTE is listening).

Configuring high-speed external caching through the Web server:

IBM HTTP Server for Windows 2000 and Windows 2003 operating systems contains a high-speed cache referred to as the *Fast Response Cache Accelerator*, or *cache accelerator*. The Fast Response Cache Accelerator is available on Windows 2000 and Windows 2003 operating systems and AIX platforms. However, support to cache dynamic content is only available on Windows 2000 and Windows 2003 operating systems. You can enable cache accelerator to cache static and dynamic content.

Before you begin

About this task

Enable cache accelerator for caching static content by adding the following directives to the `httpd.conf` configuration file, located in the IBM HTTP Server `conf` directory:

- `AfpaEnable`
- `AfpaCache on`
- `AfpaLogFile "app_server_root\IBMHttpServer\logs\afpalog" V-ECLF`

To enable cache accelerator for caching dynamic content, such as servlets and JavaServer Pages (JSP) files, configure WebSphere Application Server and IBM HTTP Server for distributed platforms:

1. Configure WebSphere Application Server to enable Fast Response Cache Accelerator.
   a. Configure an external cache group on the application server:
      1) Click **Servers > Application servers >** *server_name* **> Container services > Dynamic cache service > External cache groups**.
      2) Click **New** on the External cache group administrative console page to define an external cache group named `afpa` for each application server that uses the cache accelerator.
      3) In the **External cache group** field, type `afpa` and apply the changes.

b. Add a member to the group with an adapter bean name of com.ibm.ws.cache.servlet.Afpa.
   1) Click **Afpa > External cache group members**.
   2) Click **New** on the External cache group members administrative console page.
   3) In the **AdapterBean name** field, type com.ibm.ws.cache.servlet.Afpa.
   4) In the **Address** field, enter an unused port number.

c. Add a cache policy in the cachespec.xml file for the servlet or JSP file you want to cache. Add the following property to the cache policy:

```
<property name="ExternalCache">afpa</property>
```

2. Enable cache accelerator on IBM HTTP Server for distributed platforms:
   a. Add the following directives to the end of the httpd.conf file:
      - AfpaEnable
      - AfpaCache on
      - AfpaLogFile *"app_server_root*\IBMHttpServer\logs\afpalog" V-ECLF
      - **IBM HTTP Server 1.3.x -** LoadModule afpaplugin_module *app_server_root*\bin\afpaplugin.dll
      - **IBM HTTP Server 2.0 -** LoadModule afpaplugin_20_module *app_server_root*\bin\ afpaplugin_20.dll
      - AfpaPluginHost *WAS_Hostname:port*, where *WAS_Hostname* is the host name of the application server and *port* is the port you specified in the Address field while configuring the external cache group member

   The LoadModule directive loads the IBM HTTP Server plug-in that connects the Fast Response Cache Accelerator to the WebSphere Application Server fragment cache. If multiple IBM HTTP Servers are routing requests to a single application server, add the directives above to the httpd.conf file of each of these IBM HTTP Servers for distributed platforms.

Configuring cacheable objects with the cachespec.xml file

Use this task to define cacheable objects inside the cachespec.xml, found inside the Web module WEB-INF or enterprise bean META-INF directory.

Before you begin

Enable the dynamic cache. See “Using the dynamic cache service” on page 2210 for more information.

About this task

You can save a global cachespec.xml in the application server properties directory, but the recommended method is to place the cache configuration file with the deployment module. The root element of the cachespec.xml file is <cache>, which contains <cache-entry> elements.

The <cache-entry> element can be nested within the <cache> element or a <cache-instance> element. The <cache-entry> elements that are nested within the <cache> element are cached in the default cache instance. Any <cache-entry> elements that are in the <cache-instance> element are cached in the instance that is specified in the **name** attribute on the <cache-instance> element.

Within a <cache-entry> element are parameters that allow you to complete the following tasks to enable the dynamic cache with the cachespec.xml file:

1. Develop a cachespec.xml file.
   a. Create a caching configuration file.

      In the <*app_server_root*>/properties directory, locate the cachespec.sample.xml file.

   b. Copy the cachespec.sample.xml file to cachespec.xml in Web module WEB-INF or enterprise bean META-INF directory.
2. Define the cache-entry elements necessary to identify the cacheable objects. See the topic “cachespec.xml file” on page 2233 for a list of elements.
3. Develop cache ID rules.

To cache an object, WebSphere Application Server must know how to generate unique IDs for different invocations of that object. The *<cache-id>* element performs that task. Each cache entry can have multiple cache-ID rules that run in order until either a rule returns cache-ID that is not empty or no more rules remain to run. If no cache-ID generation rules produce a valid cache ID, then the object is not cached. Develop the cache IDs in one of two ways:

- Use the *<component>* element defined in the cache policy of a cache entry (recommended). See "cachespec.xml file" on page 2233 for more information about the *<component>* element.
- Write custom Java code to build the ID from input variables and system state. To configure the cache entry to use the ID generator, specify your IdGenerator in the XML file by using the *<idgenerator>* tag, for example:

```
<cache-entry>
     <class>servlet</class>
     <name>/servlet/CommandProcessor</name>
  <cache-id>
      <idgenerator>com.mycompany.SampleIdGeneratorImpl</idgenerator>
      <timeout>60</timeout>
  </cache-id>
</cache-entry>
```

4. Specify dependency ID rules. Use dependency ID elements to specify additional cache group identifiers that associate multiple cache entries to the same group identifier.

   The dependency ID is generated by concatenating the dependency ID base string with the values returned by its component elements. If a required component returns a null value, then the entire dependency ID does not generate and is not used. You can validate the dependency IDs explicitly through the dynamic cache API, or use another cache-entry *<invalidation>* element. Multiple dependency ID rules can exist per cache entry. All dependency ID rules run separately. See "cachespec.xml file" on page 2233 for a list of *<component>* elements.
5. Invalidate other cache entries as a side effect of this object start, if relevant. You can define invalidation rules in exactly the same manner as dependency IDs. However, the IDs that are generated by invalidation rules are used to invalidate cache entries that have those same dependency IDs.

   The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules run separately.
6. Ensure your cache policy is working correctly. You can modify the policies within the `cachespec.xml` file while your application is running. The dynamic cache reloads the updated file automatically. If you are caching static content and you are adding the cache policy to an application for the first time, you must restart the application. You do not need to restart the application server to activate the new cache policy. See "Verifying the cacheable page" on page 2233 for more information.

What to do next

Typically you declare several *<cache-entry>* elements inside a `cachespec.xml` file.

When new versions of the `cachespec.xml` are detected, the old policies are replaced. Objects that cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through its replacement algorithm.

For each of the three IDs (cache, dependency, invalidation) generated by cache entries, a *<cache-entry>* can contain multiple elements. The dynamic cache runs the *<cache-id>* rules in order, and the first one that successfully generates an ID is used to cache that output. If the object is to be cached, each one of the *<dependency-id>* elements is run to build a set of dependency IDs for that cache entry. Finally, each of the *<invalidation>* elements are run, building a list of IDs that the dynamic cache invalidates, whether or not this object is cached.

Verifying the cacheable page

Use this task to verify that the dynamic cache service has its cache policies configured correctly and is serving cached content.

Before you begin

The dynamic cache service should be enabled. You should have a cache policy developed for your application. See “Configuring cacheable objects with the cachespec.xml file” on page 2231 for more information. You must have servlet caching enabled in the web container. See “Configuring servlet caching” on page 2215 for more information.

About this task

You can verify the cacheable page by invoking the snoop servlet in the default application. If the dynamic cache is working correctly, refreshing the servlet repeatedly results in viewing cached content.

1. View the Snoop servlet in the default application by accessing the URI: `/snoop` The Snoop servlet is a part of the default application. See “Default Application” on page 40 for more information.
2. Invoke and reload the URI several times using a different Web browser or using different parameters. This action returns the same output for the snoop servlet. The snoop servlet is now operating incorrectly, because it displays the request information from its first invocation rather than from the current request.
3. Inspect the entry in the cache with the dynamic cache monitor. See “Displaying cache information” on page 2272 for more information.

cachespec.xml file

The cache parses the `cachespec.xml` file when the server starts, and extracts a set of configuration parameters from each cache-entry element. Every time a new servlet or other cacheable object initializes, the cache attempts to match each of the cache-entry elements to find the configuration information for that object.

The cache-entry elements can be inside the root cache element or inside a cache-instance element. Cache entries that are in the root element are cached with the default cache instance. Cache entries that are in the <cache-instance> element are cached in that particular cache instance. Different cacheable objects have different class elements. You can define the specific object that a cache policy refers to using the name element.

Location

Place the `cachespec.xml` file with the deployment module. Use an assembly tool to define the cacheable objects. See Assembling applications for more information about assembling applications. You can also place a global `cachespec.xml` file in the application server properties directory.

The `cachespec.dtd` file is available in the application server properties directory. The `cachespec.dtd` file defines the legal structure and the elements that can be in your `cachespec.xml` file.

Usage notes

Cachespec.xml elements

The root element of the `cachespec.xml` file is cache and contains cache-instance and cache-entry elements. The cache-entry elements can also be placed inside of cache-instance elements to make that cache entry part of a cache instance that is different from the default.

cache-instance

```
<cache-instance name="cache_instance_name"></cache-instance>
```

The name attribute is the Java Naming and Directory Interface (JNDI) name of the cache instance that is set in the administrative console.

Each cache-instance element must contain at least one cache-entry element. A cache entry that is matched within a cache-instance element is cached in the servlet cache instance that is specified by the name attribute. If identical cache-entry elements exist across cache-instance elements, the first cache-entry element that is matched is used.

cache-entry

Each cache entry must specify certain basic information that the dynamic cache uses to process that entry. This section explains the function of each cache entry element of the `cachespec.xml` file including:
- class
- name
- sharing-policy
- skip-cache
- property
- cache-id

With the current version of WebSphere Application Server, you can define multiple cache policies for a single servlet. For example, if you define multiple mappings for a servlet in the `web.xml` file, you can create a cache entry for each one of the mappings.

class

```
<class>command | servlet | webservice | JAXRPCClient | static | portlet </class>
```

This element is required and specifies how the application server interprets the remaining cache policy definition. The value `servlet` refers to servlets and JavaServer Pages (JSP) files that are deployed in the WebSphere Application Server servlet engine. The `webservice` class extends the servlet with special component types for Web services requests. The `JAXRPCClient` is used to define a cache entry for the Web services client cache. The value, `command`, refers to classes using the WebSphere Application Server command programming model. The value, `static`, refers to files that contain static content. The following examples illustrate the class element:

```
<class>command</class>
<class>servlet</class>
<class>webservice</class>
<class>JAXRPCClient</class>
<class>static</class>
<class>portlet</class>
```

name

```
<name>name</name>
```

Use the following guidelines for the name element to specify a cacheable object:
- For commands, this required element must include the package name, if any, and class name, including a trailing `.class`, of the configured object.
- For servlets and JSP files, if the `cachespec.xml` file is in the WebSphere Application Server properties directory, this required element must include the full URI of the JSP file or servlet to cache. For servlets and JSP files, if the `cachespec.xml` file is in the Web application, this required element can be relative to the specific Web application context root.
- For Web services, include the Universal Resource Identifier (URI) of the Simple Object Access Protocol (SOAP) router that is associated with the Web service that you want to cache.

- For Web services client cache, the name is the target end point of the cacheable Web service or the URI of the SOAP router that is associated with the cacheable Web service. You can use the SOAP address location in the Web Services Description Language (WSDL) file to define the name for the Web services client cache.
- For static files, if the `cachespec.xml` file is in the WebSphere Application Server properties directory, this required element must include the full URI of the file to cache. If the `cachespec.xml` file is in the Web application, this required element can be relative to the specific Web application context root. For a Web application with a context root, the cache policy for files using the static class must be specified in the Web application, and not in the properties directory.
- For portlets, if the `cachespec.xml` file is in the WebSphere Application Server properties directory, this required element must include the full context path and name of the portlet to cache. If the cachespec.xml file is in the Web application, this required element is the portlet name that is relative to the specific Web application context root.

Note: The preferred location of the `cachespec.xml` file is in the Web application, not the properties directory.

You can specify multiple name elements within a cache-entry if you have different mappings that refer to the same servlet.

The following examples illustrate the name element:

```
<name>com.mycompany.MyCommand.class</name>
<name>default_host:/servlet/snoop</name>
<name>com.mycompany.beans.MyJavaBean</name>
<name>mywebapp/myjsp.jsp</name>
<name>/soap/servlet/soaprouter</name>
<name>http://remotecompany.com:9080/service/getquote</name>
<name>mywebapp/myLogo.gif</name>
```

sharing-policy

```
<sharing-policy> not-shared | shared-push | shared-pull | shared-push-pull</sharing-policy>
```

When working within a cluster with a distributed cache, these values determine the sharing characteristics of entries that are created from this object. If this element is not present, a not-shared value is assumed. When enabling a replication, the default value is `not-shared` . This property does not affect distribution to Edge Side Include processors through the Edge fragment caching property.

Value	Description
not-shared	Cache entries for this object are not shared among different application servers. These entries can contain non-serializable data. For example, a cached servlet can place non-serializable objects into the request attributes, if the <class> type supports it.
shared-push	Cache entries for this object are automatically distributed to the dynamic caches in other application servers or cooperating Java virtual machines (JVMs). Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data.

shared-pull	Cache entries for this object are shared between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
shared-push-pull	Cache entries for this object are shared between application servers on demand. When an application server generates a cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID. On a given request for that entry, the application server knows whether to generate the entry or pull it from somewhere else. These entries cannot store non-serializable data.

The following example shows a sharing policy:

```
<sharing-policy>not-shared</sharing-policy>
```

skip-cache

Takes the name of a request attribute, which if present in the request context, dictates that the response cannot be retrieved from the cache instance that is specified. This property is useful for previewing content in production systems and verifying that the application is working and performing as expected.

```
<cache>
 <skip-cache-attribute>att1</skip-cache-attribute> <!—Applies only to the base cache- ->
...
<cache-instance name="instance1">
<skip-cache-attribute>att2</skip-cache-attribute> <!-Applies only to this instance- ->
...
</cache-instance>
</cache>
```

property

```
<property name="key">value</property>
```

where *key* is the name of the property for this cache entry element, and *value* is the corresponding value.

You can set optional properties on a cacheable object, such as a description of the configured servlet. The class determines valid properties of the cache entry. At this time, the following properties are defined:

Property	Valid classes	Value
ApplicationName	All	Overrides the J2EEName application ID so that multiple applications can share a common cache ID namespace.
EdgeCacheable	Servlet	True or false. The default is `false`. If the property is `true`, then the given servlet or JSP file is externally requested from an Edge Side Include processor. Whether or not the servlet or JSP file is cacheable depends on the rest of the cache specification.

ExternalCache	Servlet and portlet	Specifies the external cache name. The external cache name needs to match the external cache group name.
consume-subfragments	Servlet, Web service, or portlet	True or false. The default is `false`. When a servlet is cached, only the content of that servlet is stored, and includes placeholders for any other fragments to which it includes or forwards. Consume-subfragments (CSF) tells the cache not to stop saving content when it includes a child servlet. The parent entry, the one marked CSF, includes all the content from all fragments in its cache entry, resulting in one big cache entry that has no includes or forwards, but the content from the whole tree of entries. Consume-subfragments can save a significant amount of application server processing, but is typically only useful when the external HTTP request contains all the information needed to determine the entire tree of included fragments. Use the <exclude> element to tell the cache to stop consuming for the excluded fragment and instead, create a placeholder for the include or forward. For example, exclude `A.jsp` from the consume-subfragment, as follows: `<property name= "consume-sbufragments">true <exclude>/A.jsp<exclude> </property>`
do-not-consume	Servlet, Web service, or portlet	True or false. The default is `false`. When a fragment parent has the consume-subfragment property set to true the child fragment content is saved in the cache entry of the parent. Do-not-consume (DNC) tells the cache to stop saving the content for this fragment in the parent cache-entry and create a placeholder instead for the include or forward.
alternate_url	Servlet	Specifies the alternate URL that is used to invoke the servlet or JSP file. The property is valid only if the EdgeCacheable property also is set for the cache entry.
persist-to-disk	All	True or false. The default is `true`. When this property is set to `false`, the cache entry is not written to the disk when overflow or server stopping occurs.

save-attributes	Servlet and portlet	True or false. The default is `true`. When this property is set to `false`, the request attributes are not saved with the cache entry. Use the <exclude> element to specify the request attributes that do not apply to the save-attributes property. For example, to save only the attr1 attribute with the cache entry: `<property name= "save-attributes">false <exclude>attr1</exclude> </property>` To save all attributes except the `attr1` attribute in the cache entry, set the property to true in the preceding sample. If you do not use the <exclude> element, either all or no request attributes are saved with the cache entry.
delay-invalidations	Command	True or false. When this property is set to `true`, the commands that are invalidating cached objects based on the invalidation rules in this cache entry invalidate the cache entries after running. By default, the invalidation occurs before the command runs.

store-cookies	Servlet and portlet	Takes one or more cookie name as its argument which is saved along with the cache object and restored by the servlet cache in the response with a set-cookie header.
		
		Save all cookies except cookie1 as part of the cache-entry as follows:
		
		`<property name= "store-cookies">true <exclude>cookie</exclude> </property>`
		
		Save only cookie1 as part of the cache-entry, as follows:
		
		`<property name= "store-cookies">false <exclude><cookie1</exclude> </property>`

ignore-get-post	Servlet and portlet	True or false. The default is `false`. When the property is set to `true` the request type is not appended to the cache-id for GET and POST requests unless the requestType component requestType component subelement is defined. By default the request type is automatically appended to the cache-id for GET and POST requests.

do-not-cache	Servlet and portlet	Defines a fragment that is neither cached nor consumed by its parent.
		`<cache-entry>`
		`...`
		`<property name="do-not-cache">`
		`true</property>`
		
		or
		`<cache-id>`
		`<property name="do-not-cache">`
		`true</property>`
		`</cache-id>`
		`</cache-entry>`

cache-id

To cache an object, the application server must know how to generate a unique ID for different invocations of that object. These IDs are built either from user-written custom Java code or from rules that are defined in the cache policy of each cache entry. Each cache entry can have multiple cache ID rules that run in order until either:
- A rule returns a non-empty cache ID, or
- No more rules are left to run.

If none of the cache ID generation rules produce a valid cache ID, the object is not cached.

Each cache-id element defines a rule for caching an object and is composed of the sub-elements component, timeout, inactivity, priority, property, idgenerator, and metadatagenerator. The following example illustrates a cache-id element:

```
<cache-id>
   component*| timeout? | inactivity? | priority? | property* | idgenerator? | metadatagenerator?
</cache-id>
```

component subelement

Use the component subelement to generate a portion of the cache ID. The component subelement consists of the attributes id, type, and ignore-value, and the elements index, method, field, required, value, and not-value.
- Use the id attribute to identify the component.
- Use the type attribute to identify the type of component. The following table lists the values for the type.

Type	Valid classes	Meaning
method	Command	Calls the indicated method on the command or object
field	Command	Retrieves the named field in the command or object
parameter	Servlet and portlet	Retrieves the named parameter value from the request object
parameter-list	Servlet and portlet	Retrieves a list of values for the named parameter
session	Servlet and portlet	Retrieves the named value from the HTTP session
cookie	Servlet	Retrieves the named cookie value
attribute	Servlet and portlet	Retrieves the named request attribute
header	Servlet, Web service, and portlet	Retrieves the named request header

Type	Valid classes	Meaning
pathInfo	Servlet	Retrieves the pathInfo element from the request
servletpath	Servlet	Retrieves the servlet path
locale	Servlet and portlet	Retrieves the request locale
requestType	Servlet and portlet	Retrieves the HTTP request method from the request.
tiles_attribute	Servlet and portlet	Retrieves the value of an attribute from a tile.
SOAPEnvelope	Web service and Web services client cache	Retrieves the SOAPEnvelope element from a Web services request. An ID attribute of Hash uses a Hash of the SOAPEnvelope element, while Literal uses the SOAPEnvelope element as received.
SOAPAction	Web service	Retrieves the SOAPAction header, if available, for a Web services request.
serviceOperation	Web service	Retrieves the service operation for a Web services request
serviceOperationParameter	Web service	Retrieves the specified parameter from a Web services request
operation	Web services client cache	Indicates an operation type in the Web Services Description Language (WSDL) file. The id attribute is ignored and the value is the operation or method name. If the namespace of the operation is specified, format the value as `namespaceOfOperation:nameOfOperation`
part	Web services client cache	Indicates an input message part in the WSDL file or a request parameter. Its id attribute is the part or parameter name, and the value is the part or parameter value.

SOAPHeaderEntry	Web services client cache	Retrieves special information in the Simple Object Access Protocol (SOAP) header of the Web services request. The id attribute specifies the name of the entry. In addition, the entry of the SOAP header in the SOAP request must have the actor attribute, which contains com.ibm.websphere.cache. For example:
		`<soapenv:Header>`
		`<getQuote soapenv:actor=`
		`"com.ibm.websphere.cache">IBM`
		`</getQuote>`
		`</soapenv:Header>`

portletSession	Portlet	Retrieves the named value from the portlet session
portletWindowId	Portlet	Retrieves the portlet window ID from the portlet request object

Type	Valid classes	Meaning
portletMode	Portlet	Retrieves the portlet mode from the portlet request object
portletWindowsState	Portlet	Retrieves the portlet window state from the portlet request object
sessionID	Servlet and portlet	Retrieves the HTTP session ID

- Use the ignore-value attribute to specify whether or not to use the value that is returned by this component in cache ID formation. This attribute is optional with a default value of `false`. If the value is `true`, only the ID of the component is used when creating a cache ID, or no output is used when creating a dependency or invalidation ID.
- Use the method element to call a void method on a returned object. You can infinitely nest method and field objects in any combination. The method must be public and is not valid for edge-cacheable components. For example:

  ```
  <component id="getUser" type="method"><method>getUserInfo
  <method>getName</method></method></component>
  ```

  This method is equivalent to getUser().getUserInfo().getName()

  For component types attribute, method, or field that can return an object, when the object returned is a collection or array, the ID is created with a comma separated list of the elements in the collection or array. For example, if the request attribute users returns an array [a, b] and the cache entry is defined like the following example:

  ```
  <cache-entry>
   <class>servlet</class>
   <name>xxx.jsp</name>
   <cache-id>
    .
    .
    <component id="users" type="attribute">
     <required>true</required>
    </component>
    .
    .
   </cache-id>
   <dependency-id>dep
    <component id="users" type="attribute">
     <required>true</required>
    </component>
   </dependency-id>
  </cache-entry>
  ```

  The cache id contains the string `users: a,b`. The dependency id is `dep: a,b`.

  Use the multipleIDs attribute with the component types to specify and generate multiple dependency IDs (or invalidation IDs), based on the items in the collection or array. For example:

  ```
  <cache-entry>
   <class>servlet</class>
   <name>xxx.jsp</name>
   <cache-id>
    .
    .
     <component id="users" type="attribute">
      <required>true</required>
     </component>
    .
    .
   </cache-id>
   <dependency-id>dep
    <component id="users" type="attribute" multipleIDs="true">
  ```

```
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

The cache policy will generates the following dependency IDs:

- dep:a,b
- dep:a
- dep:b

Use the index element with the prvious component type to add only the value of the element at the specified index position in the collection or array, to the ID that is being created.

```
<cache-entry>
 <class>servlet</class>
 <name>xxx.jsp</name>
 <cache-id>
  .
  .
   <component id="users" type="attribute">
    <required>true</required>
    <index>1</index>
   </component>
  .
  .
 </cache-id>
 <dependency-id>dep
  <component id="users" type="attribute" multipleIDs="true">
   <required>true</required>
  </component>
 </dependency-id>
</cache-entry>
```

The previous cache policy generates the following component to use in the cache ID: `users: b`. Use the <method> element to call a void method on a returned object.

- Use the field element to access a field in a returned object. You can infinitely nest method and field objects in any combination. The field must be public. This field is not valid for edge-cacheable components. For example:

  ```
  <component id="getUser" type="method"><method>getUserInfo
  <field>name</field></method></component>
  ```

  This method is equivalent to the getUser().getUserInfo().name method.
- Use the required element to specify whether or not this component must return a non-null value for this cache ID to represent a valid cache. If set to `true`, this component must return a non-null value for this cache ID to represent a valid cache ID. If set to `false`, the default, a non-null value is used in the formation of the cache ID and a null value means that this component is not used at all in the ID formation. For example:

  ```
  <required>true</required>
  ```
- Use the value element to specify values that must match to use this component in cache ID formation. For example:

  ```
  <component id="getUser" type="method"><value>blue</value>
  <value>red</value> </component>
  ```
- Use the not-value element to specify values that must not match to use this component in cache ID formation. This method is similar to value element, but instead prescribes the defined values from caching. You can use multiple not-value elements when more than one value that is not valid exists. For example:

  ```
  <component id="getUser" type="method">
  <required>true</required>
  <not-value>blue</not-value>
  <not-value>red</not-value></component>
  ```

The component subelement can have either a method and a field element, a value element, or a not-value element. The method and field elements apply to commands only. The following example illustrates the attributes of a component sub-element:

```
<component id="isValid" type="method" ignore-value="true"><component>
```

timeout subelement

The timeout subelement is used to specify an absolute time-to-live (TTL) value for the cache entry. For example,

```
<timeout>value</timeout>
```

where *value* is the amount of time, in seconds, to keep the cache entry. Cache entries that are in memory are kept indefinitely, as long as the entries remain in memory. Cache entries that are stored on disk are evicted if they are not accessed for 24 hours.

inactivity subelement

The inactivity subelement is used to specify a time-to-live (TTL) value for the cache entry based on the last time that the cache entry was accessed. It is a subelement of the cache-id element.

```
<inactivity>value</inactivity>
```

where *value* is the amount of time, in seconds, to keep the cache entry in the cache after the last cache hit.

priority subelement

Use the priority subelement to specify the priority of a cache entry in a cache. The priority weighting is used by the least recently used (LRU) algorithm of the cache to decide which entries to remove from the cache if the cache runs out of storage space. For example,

```
<priority>value</priority>
```

where *value* is a positive integer between 1 and 16 inclusive.

Samples

The following sample keeps the cache entry in the cache for a minimum of 35 seconds and a maximum of 180 seconds. If the cache entry is accessed within each 35 second inactivity period, the inactivity period is extended for another 35 seconds. However, because the timeout element is also configured, the cache entry is always invalidated after 180 seconds. If the cache entry is not accessed within the 35 second period, the entry is removed from the cache.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
 </component>
 <timeout>180</timeout>
 <inactivity>35</inactivity>
 <priority>1</priority>
</cache-id>
```

The following sample keeps the cache entry in the cache for a minimum of 600 seconds. If the cache entry is accessed within each 600 second period, the inactivity period is extended for another 600 seconds. If the cache entry is not accessed within the 600 second period, the cache entry is removed from the cache.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
```

```
 </component>
 <inactivity>600</inactivity>
 <priority>1</priority>
</cache-id>
```

In the following sample, the value for inactivity has no meaning because the timeout period is less than the inactivity period. The cache entry is always invalidated after 180 seconds, no matter how often the cache entry is accessed.

```
<cache-id>
 <component id="timeout" type="parameter">
  <required>true</required>
 </component>
 <timeout>180</timeout>
 <inactivity>600</inactivity>
 <priority>1</priority>
</cache-id>
```

property subelement

Use the property subelement to specify generic properties for the cache entry. For example,

```
<property name="key">value</property>
```

where *key* is the name of the property to define, and *value* is the corresponding value.

For example:

```
<property name="description">The Snoop Servlet</property>
```

Property	Valid classes	Meaning
sharing-policy/timeout/priority	All	Overrides the settings for the containing cache entry when the request matches this cache ID.
EdgeCacheable	Servlet	Overrides the settings for the containing cache entry when the request matches this cache ID.

idgenerator and metadatagenerator sub-elements

Use the idgenerator element to specify the class name that is loaded for the generation of the cache ID. The IdGenerator element must implement the com.ibm.websphere.servlet.cache.IdGenerator interface for a servlet or the com.ibm.websphere.webservices.IdGenerator interface for the Web services client cache. An example of the idgenerator element follows:

```
<idgenerator> class name </idgenerator>
```

Where `class name` is the fully-qualified name of the class to use. Define this generator class in a shared library.

Use the metadatagenerator element inside the cache-id element to specify the class name loaded for the metadata generation. The MetadataGenerator class must implement the com.ibm.websphere.servlet.cache.MetaDataGenerator interface for a servlet or the com.ibm.websphere.cache.webservices.MetaDataGenerator interface for Web services client cache. The MetadataGenerator class defines properties like timeout, inactivity, external caching properties or dependencies. An example of the `metadatagenerator` element follows:

```
<metadatagenerator> classname </metadatagenerator>
```

In this example, class name is the fully-qualified name of the class to use. Define this generator class in a shared library.

dependency-id element

Use the dependency-id element to specify additional cache identifiers that associate multiple cache entries to the same group identifier.

The value of the dependency-id element is generated by concatenating the dependency ID base string with the values that are returned by its component elements. If a required component returns a null value, the entire dependency does not generate and is not used. Validate the dependency IDs explicitly through the dynamic cache API, or use the invalidation element. Multiple dependency ID rules can exist in one cache-entry element. All dependency rules run separately.

invalidation element

To invalidate cached objects, the application server must generate unique invalidation IDs. Build invalidation IDs by writing custom Java code or through rules that are defined in the cache policy of each cache entry. The following example illustrates an invalidation in the cache policy:

```
<invalidation>component* | invalidationgenerator? </invalidation>
```

invalidationgenerator subelement

The invalidationgenerator element is used with the Web Services client cache only. Use the invalidationgenerator element to specify the class name to load for generating invalidation IDs. The InvalidationGenerator class must implement the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. An example of the invalidationgenerator element follows:

```
<invalidationgenerator>class name</invalidationgenerator>
```

In this example, classname is the fully qualified name of the class that implements the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. Define this generator class in a shared library.

Configuring command caching

Cacheable commands are stored in the cache for reuse with a similar mechanism for servlets and JavaServer Pages (JSP) files.

About this task

In this case, however, the unique cache IDs are generated based on methods and fields present in the command as input parameters. For example, a **GetStockQuote** command can have a symbol as its input parameter.

A unique cache ID can generate from the name of the command, plus the value of the symbol.

To use command caching you must:

Create a command.

1. Define an interface. The Command interface specifies the most basic aspects of a command.

   You must define the interface that extends one or more of the interfaces in the command package. The command package consists of three interfaces:
   - TargetableCommand
   - CompensableCommand
   - CacheableCommand

   In practice, most commands implement the TargetableCommand interface, which allows the command to run remotely. The code structure of a command interface for a targetable command follows:

```
...
import com.ibm.websphere.command.*;
public interface MyCommand extends TargetableCommand {
      // Declare application methods here
}
```

2. Provide an implementation class for the interface. Write an interface that extends the CacheableCommandImpl class and implements your command interface. This class contains the code for the methods in your interface, the methods inherited from extended interfaces like the CacheableCommand interface, and the required or abstract methods in the CacheableCommandImpl class.

   You can also override the default implementations of other methods provided in the CacheableCommandImpl class.

Command class

Extend one or more of the three interfaces included in the command package to write a command interface. The base interface for all commands is the Command interface.

The Command interface provides only the client-side interface for generic commands and declares three basic methods:

- **isReadyToCallExecute.** This method is called on the client side before the command runs on server.
- **execute.** This method passes the command to the target and returns any data.
- **reset.** This method reverts any output properties to the values they had before the execute method was called so that you can reuse the object.

The implementation class for your interface must contain implementations for the isReadyToCallExecute and reset methods.

CacheableCommandImpl class

Commands are implemented by extending the class CacheableCommandImpl, which implements the CacheableCommand interface.

The CacheableCommandImpl class is an abstract class that provides implementations for some of the methods in the CacheableCommand interface, for example, setting return values. This class declares additional methods that the application must implement, for example, how to run the command.

The code structure of an implementation class for the CacheableCommand interface follows:

```
...
import com.ibm.websphere.command.*;
public class MyCommandImpl extends CacheableCommandImpl
implements MyCommand {
// Set instance variables here      ...
// Implement methods in the MyCommand interface      ...
// Implement abstract methods in the CacheableCommandImpl class
...
}
```

Example: Caching a command object

Cacheable commands are stored in the cache for reuse with a similar mechanism for servlets and JavaServer Pages (JSP) files.

This example of command caching is a simple stock quote command.

The following is a stock quote command bean. It accepts a ticker as an input parameter and produces a price as its output parameter.

```
public class QuoteCommand extends CacheableCommandImpl
{
    private String ticker;
    private double price;
```

```
    // called to validate that command input parameters have been set
    public boolean isReadyToCallExecute() {
      return (ticker!=null);
    }
    // called by a cache-hit to copy output properties to this object
    public void setOutputProperties(TargetableCommand fromCommand) {
        QuoteCommand f = (QuoteCommand)fromCommand;
        this.price = f.price;
    }

   // business logic method called when the stock price must be retrieved
    public void performExecute()throws Exception {...}

    //input parameters for the command
    public void setTicker(String ticker) { this.ticker=ticker;}
    public String getTicker() { return ticker;}

    //output parameters for the command
    public double getPrice()  { return price;};
}
```

To cache the above command object using the stock ticker as the cache key and using a 60 second time-to-live, use the following cache policy:

```
<cache>
 <cache-entry>
  <class>command</class>
  <sharing-policy>not-shared</sharing-policy>
  <name>QuoteCommand</name>
  <cache-id>
   <component type="method" id="getTicker">
    <required>true</required>
   </component>
   <priority>3</priority>
   <timeout>60</timeout>
  </cache-id>
 </cache-entry>
</cache>
```

Configuring the Web services client cache

The Web services client cache is a part of the dynamic cache service that is used to increase the performance of Web services clients by caching responses from remote Web services. Configuring the Web services client cache can improve the performance of your application server by caching the responses from remote Web services for a specified amount of time.

Before you begin

Enable the dynamic cache service. To enable the dynamic cache service, see “Task overview: Using the dynamic cache service to improve performance” on page 2197. Before attempting to configure the Web services client cache, understand how to create basic cache policies. See “Configuring cacheable objects with the cachespec.xml file” on page 2231 for more information.

About this task

Enabling the Web services client cache is an option to improve the performance of your system by using the dynamic cache service to save responses from remote Web services for a specified amount of time. After a response is returned from a remote Web service, the response is saved in the client cache on the application server. Any identical requests that are made to the same remote Web service are then responded to from the cache for a specified period of time. The Web services client cache relies primarily on time-based invalidations because the target web service can be outside of your enterprise network and unaware of your client caching. Therefore, you can specify the amount of time in the cache and the rules to build cache entry IDs in the cache in your client application.

The Web services client cache is provided as a Java API for XML-Based Remote Procedure Calls (JAX-RPC) handler on your application server. This JAX-RPC cache handler intercepts the SOAP requests that flow through it from application clients. It then identifies a cache policy based on the target Web service. After a policy is found, all the cache ID rules are evaluated one by one until a valid rule is detected.

1. Locate the Web Services Description Language (WSDL) file for the remote service. Portions of the WSDL file contain information that you will use in writing your cache policy. For more information about WSDL files, see "WSDL" on page 338. Following is an example of portions of a WSDL file that contains values that are used for the purpose of demonstration.

```
<definitions targetNamespace="http://TradeSample.com/"
  xmlns:tns="http://TradeSample.com/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <message name="getQuoteRequest">
  <part name="symbol" type="xsd:string"/>
 </message>
.....
.....
<binding name="SoapBinding" type="tns:GetQuote">
 <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getQuote">
   <soap:operation soapAction=""/>
   <input name="getQuoteRequest">
    <soap:body namespace="http://TradeSample.com/"
    use="encoded"
   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   </input>
   ......
   </operation>
</binding>
<service name="GetQuoteService">
 <port binding="tns:SoapBinding" name="SoapPort">
  <soap:address location="http://TradeSample.com:9080/service/getquote"/>
 </port>
</service>
</definitions>
```

   The highlighted text indicates values that are used in writing your cache policy.

2. Choose how you plan to generate the cache id for your Web services client caching. You can build your cache id rules by using one of four options:
   - By calculating a hash of the SOAPEnvelope
   - By using SOAPHeader entries
   - By using operation and part parameters
   - By using custom Java code to build the cache id from input SOAP message content

   Using SOAPHeader entries is the best option if you can include information for building cache keys as part of the SOAP header. This method creates easy to read cache keys and can be built without parsing the SOAP body. Use custom Java code to generate a specific cache id based on the SOAP message. If you cannot include the header information, you can calculate the hash of the SOAPEnvelope for performance or parse the SOAP Body for user-friendly cache keys.

3. Develop your cache policy.

   All Web services client cache policies must have the **class** *JAXRPCClient*. The **name** element in each cache entry is the target endpoint location that is defined in the WSDL file. You can find this address in the WSDL file by finding the `<soap:address location="..."/>` tag located in the **port** element. In the WSDL file for this sample, the address is **http://TradeSample.com:9080/service/getquote**. Develop the rest of your cache policy by using one of the following options:
   - **Calculate a hash of the SOAPEnvelope to identify the request**

```
<cache>
 <cache-entry>
  <class>JAXRPCClient</class>
  <name>http://TradeSample.com:9080/service/getquote</name>
  <cache-id>
   <component id="hash" type="SOAPEnvelope"/>
   <timeout>60</timeout>
  </cache-id>
 </cache-entry>
</cache>
```

Note the **component** attributes to create a cache id based on a hash calculation of the SOAPEnvelope. The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:Hash=xxxHashSoapEnvelope`.

- **Use the SoapHeader to identify the request**

```
<cache>
 <cache-entry>
  <class>JAXRPCClient</class>
  <name>http://TradeSample.com:9080/service/getquote</name>
  <cache-id>
   <component id="urn:stock:getQuote" type="SOAPHeaderEntry"/>
  </cache-id>
 </cache-entry>
</cache>
```

This cache id is built by using special information in the SOAP header to identify requests for entries in the cache. Specify the **type** as `SOAPHeaderEntry` and the **id** as the `operation` name located in the **binding** element in the WSDL file. The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:urn:stock:getQuote=IBM`.

An example of a SOAP request generated by the client using SOAP Header:

Note that the **soapenv:actor** attribute must contain `com.ibm.websphere.cache`.

```
POST /wsgwsoap1/soaprpcrouther HTTP/1.1
SOAPAction: ""
Context-Type: text/xml; charset=utf-8
User-Agent: Java/1.4.1
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 645

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
 <getQuote soapenv:actor="com.ibm.websphere.cache" xmlns="urn:stock">IBM</getQuote>
</soapenv:Header>
<soapenv:Body
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
 <getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
  <symbol xsi:type="xsd:string">IBM</symbol>
 </getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

- **Use operation and part to identify the request**

```
<cache>
 <cache-entry>
  <class>JAXRPCClient</class>
  <name>http://TradeSample.com:9080/service/getquote</name>
  <cache-id>
   <component id="" type="operation">
    <value>http://TradeSample.com/:getQuote</value>
```

```
   </component>
   <component id="symbol" type="part"/>
  </cache-id>
 </cache-entry>
</cache>
```

This example uses operation and request parameters. The `operation` can be a method name in the WSDL file located in the **binding** element or a method name in the Document/Literal Invocation (DII). If the namespace of the operation is defined, the value is formatted as `namespaceOfOperation:nameOfOperation`. The `part` type can be defined in the **message** element of the WSDL file, as a request parameter, or as a request parameter of the DII invocation. Its `id` attribute is the part or parameter name, and the `value` is the part or parameter value. The cache id generated from using operation and request parameters is `http://TradeSample.com:9080/service/getquote:operation=http://TradeSample.com/:getQuote/symbol=IBM`.

An example of the SOAP request generated by the client using operation and part:

```
POST /wsgwsoap1/soaprpcrouter HTTP/1.1
SOAPAction:""
Content-Type: text/xml/charset=utf-8
User-Agent: Java/1.4.1
Host: localhost
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Current-Length: 645

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns: soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body
 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
  <symbol xsi:type="xsd:string">IBM</symbol>
 </getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

- **Use custom Java code to build the cache id from input SOAP message content**

  If you use custom Java code to build the cache id, create an ID generator Java class that implements the IdGenerator interface defined in the com.ibm.websphere.cache.webservices.IdGenerator package and add a reference to the class you create in the cachespec.xml file by using the **idgenerator** tag.

  You can also implement the com.ibm.websphere.cache.webservices.MetaDataGenerator package to assign cache metadata such as timeout, priority, and dependency ids to cache entries using the **metadatagenerator** tag.

  Implement the com.ibm.websphere.cache.webservices.InvalidationGenerator interface and use the **invalidationgenerator** tag in the cachespec.xml file to generate cache ids and to invalidate entries in the cache. The id generated by the invalidation generator can be a cache id or a dependency id.

  For example, if you develop an ID generator class named SampleIdGeneratorImpl, a metadata generator class named SampleMetaDataGeneratorImpl, and an invalidation generator class named SampleInvalidationGeneratorImpl, your cachespec.xml file might contain the following:

```
<cache-entry>
 <class>JAXRPCClient</class>
 <name>http://TradeSample.com:9080/service/getquote</name>
 <cache-id>
  <idgenerator>com.mycompany.SampleIdGeneratorImpl</idgenerator>
  <metadatagenerator>
       com.mycompany.SampleMetaDataAndInvalidationGeneratorImpl
    </metadatagenerator>
  <timeout>60</timeout>
 </cache-id>
```

```
 <invalidation>http://TradeSample.com:9080/service/GetQuote
  <invalidationgenerator>
       com.mycompany.SampleMetaDataAndInvalidationGeneratorImpl
    </invalidationgenerator>
 </invalidation>
</cache-entry>
```

The SampleIdGeneratorImpl class is a custom Java class that implements the `com.websphere.cache.webservices.IdGenerator` interface. The SampleIdGeneratorImpl class contains the getID method:

```
String getId(javax.xml.rpc.handler.soap.SOAPMessageContext messageContext)
```

The following is an example of the SampleIdGeneratorImpl.java class.

```
public class SampleIdGeneratorImpl implements IdGenerator {
//The SampleIdGenerator class builds cache keys using SOAP header entries
    public String getId(javax.xml.rpc.handler.soap.SOAPMessageContext
   messageContext) {
   ....
   // retrieve SOAP header entries from SOAPMessage
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                  (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
     if (headerEntryName.equals("getQuote")){
      String sNamespace = element.getNamespaceURI("");
      if (sNamespace != null && !sNamespace.equals("")) {
          headerEntryName = sNamespace + ":" + headerEntryName;
       String quotes = element.getValue();
      }
      ...
      ...
      // create a method "parseAndSort" to parse and sort quotes
      // By parsing and sorting quotes, you avoid duplicate cache
      // entries.
      // quotes e.g. IBM,CSCO,MSFT,INTC
      // to return a cache key "urn:stock:getQuote=CSCO,IBM,INTC,MSFT"
      String sortQuotes = parseAndSort(quotes);
      cacheKey = headerEntryName + "=" + sortQuotes;
       }
    }
  return cacheKey;
  }
 }
```

The cache id for this sample is generated as `http://TradeSample.com:9080/service/getquote:urn:stock:symbol=CSCO,IBM,INTC,MSFT`.

The SampleMetaDataAndInvalidationGeneratorImpl class is a custom Java class that implements the `com.websphere.cache.webservices.MetaDataGenerator` interface and the `com.websphere.cache.webservices.InvalidationGenerator` interface. The SampleMetaDataAndInvalidationGeneratorImpl class contains the setMetaData method and the getInvalidationIds method. You can also set up two smaller classes instead of this one large class. For example, create one class for the metadata generator and a different class for the invalidation generator. The following are method prototypes for the setMetaData method and the getInvalidationIds method:

```
void setMetaData (javax.xml.rpc.handler.soap.SOAPMessageContext messageContext,
 com.ibm.websphere.cache.webservices.JAXRPCEntryInfo entryInfo)
String[] getInvalidationIds (javax.xml.rpc.handler.soap.SOAPMessageContext messageContext)
```

An example of the SampleMetaDataAndInvalidationGeneratorImpl.java class follows:

```
public class SampleMetaDataAndInvalidationGeneratorImpl implements
  MetaDataGenerator, InvalidationGenerator {
    //assigns time limit, and priority metadata
    public void setMetadata(javax.xml.rpc.handler.soap.SOAPMessageContext messageContext,
  com.ibm.websphere.cache.webservices.JAXRPCEntryInfo entryInfo) {
   ....

  // retrieve SOAP header entries from SOAPMessage
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                 (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
                                if (headerEntryName.equals("metadata")) {
     // retrieve each metadata element and set metadata
                 entryInfo.setTimeLimit(timeLimit);
                entryInfo.setPriority(priority);
            }
        }
    }

    //builds invalidation ids using SOAP header.
    public String[] getInvalidationIds(javax.xml.rpc.handler.soap.SOAPMessageContext
      messageContext) {   ....
   // retrieve SOAP header entries from SOAPMessage
                String[] invalidationIds = new String[1];
   SOAPHeader sh = soapEnvelope.getHeader();
     if (sh != null) {
    Iterator it = sh.examineHeaderElements("com.mycompany.actor");
    while (it.hasNext()) {
       SOAPHeaderElement element =
                 (SOAPHeaderElement)it.next();
     Name name = element.getElementName();
     String headerEntryName = name.getLocalName();
                                if (headerEntryName.equals("invalidation")) {
     String sNamespace = element.getNamespaceURI("");
     if (sNamespace != null && !sNamespace.equals("")) {
         headerEntryName = sNamespace + ":symbol";
      String quotes = element.getValue();
     }
     ...
     ...
     // create a method "parseAndSort" to parse and sort quotes
     // By parsing and sorting quotes, you avoid duplicate cache
     // entries.
     // quotes e.g. SUNW,NT
     // to return a cache key "urn:stock:symbol=NT,SUNW"
     String sortQuotes = parseAndSort(quotes);
     invalidationIds[0] = headerEntryName + "=" sortQuotes;
             }
     }
  return invalidationIds;
    }
}
```

The invalidation id for this sample is generated as:

```
http://TradeSample.com:9080/service/getquote:urn:stock:symbol=NT,SUNW
```

An example of the SOAP request generated by the client when using custom Java code follows:

```
POST /wsgwsoap1/soaprpcrouter HTTP/1.1
SOAPAction: ""
Context-type: text/xml, charset=utf-8
User-Agent: Java/1.4.1
Host: localhost
```

```
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length:645

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
 <getQuote soapenv:actor="com.mycompany.actor"
   xmlns="urn:stock">IBM,CSCO,MSFT,INTC</getQuote>
  <metaData soapenv:actor="com.mycompany.actor" xmlns="urn:stock">
   <priority>10</priority>
   <timeLimit>30000</timeLimit>
  </metaData>
  <invalidation soapenv:actor="com.mycompany.actor"
   xmlns="urn:stock">SUNW, NT</invalidation>
</soapenv:Header>
<soapenv:Body
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
<getQuote xmlns="urn:ibmwsgw#GetQuoteSample">
 <symbol xsi:type="xsd:string">IBM,CSCO,MSFT,INTC</symbol>
</getQuote>
</soapenv:Body>
</soapenv:Envelope>
```

4. Save the cache policy to the appropriate directory.
   - If you are using the Web Services Gateway on SOAP channel 1, the directory is: <app_server_root>\installedApps\wsgwsoap1.*servername*.*nodename*.ear/wsgwsoap.war/WEB-INF
   - If you are using a simple JAX-RPC client in your application to invoke remote Web services, save your cache policy in the Web module WEB-INF of your JAX-RPC application.

Results

You can monitor the results of your Web services client cache policy by using the dynamic cache monitor. See “Displaying cache information” on page 2272 for more information.

Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache

By using the DistributedMap or DistributedObjectCache interfaces, Java Platform, Enterprise Edition (Java EE) applications and system components can cache and share Java objects by storing a reference to the object in the cache.

About this task

The DistributedMap and DistributedObjectCache interfaces are simple interfaces for the dynamic cache. Using these interfaces, Java EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default dynamic cache instance is created if the dynamic cache service is enabled in the administrative console. This default instance is bound to the global Java Naming and Directory Interface (JNDI) namespace using the name services/cache/distributedmap.

Multiple instances of the DistributedMap and DistributedObjectCache interfaces on the same Java virtual machine (JVM) enable applications to separately configure cache instances as needed. Each instance of the DistributedMap interface has its own properties.

Note: For more information about the DistributedMap and DistributedObjectCache interfaces, see the API documentation for the com.ibm.websphere.cache package. See Additional Application Programming Interfaces (APIs) for more information.

Note: If you are using custom object keys, you must place your classes in a shared library. You can define the shared library at cell, node, or server level. Then, in each server create a class loader and associate it with the shared library that you defined. See the *Setting up the application serving environment* PDF for more information.

Place JAR files in a shared library when you deploy the application in a cluster with replication enabled. Simply turning on replication does not require a shared library; however, if you are using application-specific Java objects, such as cache key or cache value, those Java classes are required to be in the shared library.

There are four methods for configuring and using cache instances:

- Configuring the default object cache (method one below)
- Creating and configuring the custom object cache (method three below)
- Creating and configuring the custom object cache by using the cacheinstances.properties file (method four below)
- Using the resource reference (method five below)
- **Method 1 - Configure default cache instances**. The default servlet cache instance (JNDI name: `services/cache/basecache`) is created when the server starts , if servlet caching is enabled. The default object cache instance (JNDI name: `services/cache/distributedmap`) is always created when the server starts.
  1. In the administrative console, select **Servers > Application servers >** *<server_name>* **Container services > Dynamic cache services**.
  2. Configure other cache settings. Refer to the Dynamic cache service settings article for more information.
  3. Click **Apply** or **OK**.
  4. Restart WebSphere Application Server.

  You can use the following code to look up the cache instances:

```
 InitialContext ic = new InitialContext();
DistributedMap dm1 = (DistributedMap)ic.lookup("services/cache/instance_one");

DistributedMap dm2 = (DistributedMap)ic.lookup("services/cache/instance_two");

// or

InitialContext ic = new InitialContext();
DistributedObjectCache dm1 = (DistributedObjectCache)ic.lookup("services/cache/instance_one");

DistributedObjectCache dm2 = (DistributedObjectCache)ic.lookup("services/cache/instance_two");
```

-
- **Method 2 - Configure servlet cache instances**. A servlet cache instance is a location, in addition to the default servlet cache, where dynamic cache can store, distribute, and share data. By using servlet cache instances, your applications have greater flexibility and better tuning of the cache resources. The Java™ Naming and Directory Interface (JNDI) name that is specified for the cache instance is mapped to the name attribute in the `<cache instance>` tag in the `cachespec.xml` configuration file.
  1. In the administrative console, click **Resources > Cache instances > Servlet cache instances**.
  2. Enter the scope, as follows:
     – Specify CELL SCOPE to view and configure cache instances that are available to all servers within the cell.

- Specify NODE SCOPE to view and configure cache instances that are available to all servers with the particular node.
- Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.

3. Enter the required display name for the resource in the name field.
4. Enter the JNDI name for the resource. Specify this name in the attribute field in the `<cache-instance>` tag in the `cachespec.xml` configuration file. This tag finds the particular cache instance in which to store cache entries.
5. Configure other cache settings. Refer the Dynamic cache service settings article for more information.
6. Click **Apply** or **OK**.
7. **Optional:**If you want to set up additional custom properties for this instance, click **Resources > Cache instances > Servlet cache instances** *<servlet_cache_instance_name>* **> Custom properties > New**.
8. **Optional:** Enter the name of the custom property in the Name field. Refer to the Dynamic cache custom properties article for more information.

   Note: Use the custom property with scope indicated **Per cache instnace** only. For example, enter `createCacheAtServerStartup` in the Name field.
9. Enter a valid value for the property in the Value field.
10. Save the property and restart WebSphere Application Server.

-
- **Method 3 - Configure object cache instances**. An object cache instance is a location, in addition to the default object cache, where dynamic cache can store, distribute, and share data for Java™ Platform, Enterprise Edition (Java EE) applications. Use cache instances to give applications better flexibility and tuning of the cache resources. Use the DistributedObjectCache programming interface to access the cache instances. For more information about the DistributedObjectCache application programming interface, see the API documentation.

  Note: Method three is an extension to method one or method two, listed above. First use either method one or method two.

  Create and configure the object cache instance, as follows:
  1. In the administrative console, click **Resources > Cache instances > Object cache intances**.
  2. Enter the scope:
     - Specify CELL SCOPE to view and configure cache instances that are available to all servers within the cell.
     - Specfy NODE SCOPE to view and configure cache instances that area available to all servers with the particular node.
     - Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.
  3. Enter the required display name for the resource in the name field.
  4. Enter the JNDI name for the resource. Use this name when looking up a reference to this cache instance. The results return a DistributedMap object.
  5. Configure other cache settings. See the Dynamic cache service setting article for more information.
  6. Click **Apply** or **OK**.
  7. **Optional:** If you want to set up additional custom properties for this instance, click **Resources > Cache instances > Object cache instances** *<servlet_cache_instance_name>* **Custom properties > New**.
  8. **Optional:** Enter the name of the custom property in the Name field.

Note: Use the custom property with scope indicated **Per cache instance** only. For example, enter `createCacheAtServerStartup` in the Name field.

9. Enter a valid value for the property in the Value field.
10. Save the property and restart WebSphere Application Server.

If you defined two object cache instances in the administrative console with JNDI names of **services/cache/instance_one** and **services/cache/instance_two**, you can use the following code to look up the cache instances:

```
 InitialContext ic = new InitialContext();
DistributedMap dm1 = (DistributedMap)ic.lookup("services/cache/instance_one");

DistributedMap dm2 = (DistributedMap)ic.lookup("services/cache/instance_two");

// or

InitialContext ic = new InitialContext();
DistributedObjectCache dm1 = (DistributedObjectCache)ic.lookup("services/cache/instance_one");

DistributedObjectCache dm2 = (DistributedObjectCache)ic.lookup("services/cache/instance_two");
```

- **Method 4 - Configure cache instances using the cacheinstances.properties file**. You can create cache instances using the cacheinstances.properties file and package the file in your Enterprise Archive (EAR) file. Use the table information in the cacheinstances.properties file article as a reference of the names, values, and explanations.

  The first line defines the cache instance name. The subsequent lines define custom properties. The formats are as follows:

  ```
  cache.instance.x=InstanceName
  cache.instance.x.customPropertyName=customPropertyValue
  ```

  where:

  – *x* is the instance name number, which starts with `0`.
  – *customPropertyName* is the custom property name. Refer to the Dynamic cache custom properties article for more information.

    Note: Use the custom property with scope indicated per cache instance only.

  – *customPropertyValue* is the possible custom property value.

  Examples

  The following is an example of how you can create additional cache instances using the cacheinstances.properties. file:

  ```
  cache.instance.0=/services/cache/instance_one
  cache.instance.0.cacheSize=1000
  cache.instance.0.enableDiskOffload=true
  cache.instance.0.diskOffloadLocation=${app_server_root}/diskOffload
  cache.instance.0.flushToDiskOnStop=true
  cache.instance.0.useListenerContext=true
  cache.instance.0.enableCacheReplication=false
  cache.instance.0.disableDependencyId=false
  cache.instance.0.htodCleanupFrequency=60
  cache.instance.1=/services/cache/instance_two
  cache.instance.1.cacheSize=1500
  cache.instance.1.enableDiskOffload=false
  cache.instance.1.flushToDiskOnStop=false
  cache.instance.1.useListenerContext=false
  cache.instance.1.enableCacheReplication=true
  cache.instance.1.replicationDomain=DynaCacheCluster
  cache.instance.1.disableDependencyId=true
  ```

  The preceding example creates two cache instances named instance_one and instance_two. Cache instance_one has a cache entry size of `1,000` and instance_two has a cache entry size of `1,500`. Disk offload is enabled in instance_one and disabled in instance_two. Use listener context is enabled in

instance_one and disabled in instance_two. Flush to disk on stop is enabled in instance_one and disabled in instance_two. Cache replication is enabled in instance_two and disabled in instance_one. The name of the data replication domain for instance_two is DynaCacheCluster. Dependency ID support is disabled in instance_two.

Place the cacheinstances.properties file in either your application server or application class path. For example, you can use your application WAR file, WEB-INF\classes directory or server_root\classes directory. The first entry in the properties file (cache.instance.0) specifies the JNDI name for the cache instance in the global namespace.

You can use the following code to look up the cache instance:

```
  InitialContext ic = new InitialContext();
     DistributedMap dm1 =
(DistributedMap)ic.lookup("services/cache/instance_one");
     DistributedMap dm2 =
(DistributedMap)ic.lookup("services/cache/instance_two");

// or

InitialContext ic = new InitialContext();
DistributedObjectCache dm1 =  (DistributedObjectCache)ic.lookup("services/cache/instance_one");

DistributedObjectCache dm2 = (DistributedObjectCache)ic.lookup("services/cache/instance_two");
```

- **Method 5: Resource reference**.

  Note: This method is an extension to method three and method four, listed above. First use either method three or method four.

  Define a resource-ref in your module deployment descriptor (web.xml and ibm-web-bnd.xmi files) and look up the cache using the `java:comp` namespace.

  Resource-ref example:

  File: web.xml

```
<resource-ref id="ResourceRef_1">
  <res-ref-name>dmap/LayoutCache</res-ref-name>
  <res-type>com.ibm.websphere.cache.DistributedMap</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
<resource-ref id="ResourceRef_2">
  <res-ref-name>dmap/UserCache</res-ref-name>
  <res-type>com.ibm.websphere.cache.DistributedMap</res-type>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

  File: ibm-web-bnd.xmi

```
<?xml version="1.0" encoding="UTF-8"?>
<webappbnd:WebAppBinding xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:webappbnd="webappbnd.xmi"
xmlns:webapplication="webapplication.xmi" xmlns:commonbnd="commonbnd.xmi"
xmlns:common="common.xmi"
xmi:id="WebApp_ID_Bnd" virtualHostName="default_host">
  <webapp href="WEB-INF/web.xml#WebApp_ID"/>
  <resRefBindings xmi:id="ResourceRefBinding_1"
jndiName="services/cache/instance_one">
  <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_1"/>
  </resRefBindings>
  <resRefBindings xmi:id="ResourceRefBinding_2"
jndiName="services/cache/instance_two">
  <bindingResourceRef href="WEB-INF/web.xml#ResourceRef_2"/>
  </resRefBindings>
  </webappbnd:WebAppBinding>
```

  The following example shows how to look up the resource-ref:

```
InitialContext ic = new InitialContext();
  DistributedMap dm1a =(DistributedMap)ic.lookup("java:comp/env/dmap/LayoutCache");
  DistributedMap dm2a =(DistributedMap)ic.lookup("java:comp/env/dmap/UserCache");
// or
  DistributedObjectCache dm1a =(DistributedObjectCache)ic.lookup("java:comp/env/dmap/LayoutCache");
  DistributedObjectCache dm2a =(DistributedObjectCache)ic.lookup("java:comp/env/dmap/UserCache");
```

The previous resource-ref example maps `java:comp/env/dmap/LayoutCache` to `/services/cache/instance_one` and `java:comp/env/dmap/UserCache` to `/services/cache/instance_two`. In the examples, DistributedMap dm1 and dm1a are the same object. DistributedMap dm2 and dm2a are the same object.

- **Method 6: Java virtual machine cache settings**. You can set the custom properties globally to affect all cache instances. This overwrites the settings in method 1, method 2 and method 3, but not method 4 (cacheinstances.properties). Configure the cache instance globally, as follows:
  1. In the administrative console, click **Servers > Application servers > server_name > Java and process management > Process definition > Java virtual machine > Custom properties > New**.
  2. Enter the name of the custom property in the Name field. Refer to the Dynamic cache custom properties article for more information. After you find the custom property name, add the `com.ibm.ws.cache.CacheConfig` prefix to the front of custom property name. For example, if the custom property name is `createCacheAtServerStartup`, enter `com.ibm.ws.cache.CacheConfig.createCacheAtServerStartup` in the Name field.
  3. Enter a valid value for the property in the Value field.
  4. Save the property and restart WebSphere Application Server.

Object cache instance settings

An object cache instance is a location, in addition to the default shared dynamic cache, where any Java 2 Platform, Enterprise Edition (J2EE) application can store, distribute, and share data. This gives applications greater flexibility and better tuning of the cache resources. Use the DistributedMap programming interface to access this cache instance. See the API documentation for more information.

To view this administrative console page, click **Resources > Cache instances > Object cache instances > *cache_instance_name***.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Use this name when looking up a reference to this cache instance. The results return a `DistributedMap` object.

Description:

Specifies a description for the resource. This field is optional.

Category:

Specifies a category string to classify or group the resource. This field is optional.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands.

Default	2000

Range	100 - 200,000

Default priority:

Specifies the default priority for servlets that can be cached. This value determines how long an entry stays in a full cache.

The recommended value is one. The range is one through 255.

Enable disk offload:

Specifies if disk offloading is enabled.

If you have disk offload disabled, when a new entry is created while the cache is full, the priorities are configured for each entry and the least recently used algorithm are used to remove the entry from the cache in memory. If you enable disk offload, the entry that would be removed from the cache is copied to the local file system. The location of the file is specified by the disk offload location.

Default	false

Offload location:

Specifies the directory that is used for disk offload.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/`*`node/server name`*`/_dynacache/`*`cache JNDI name`* will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/`*`node/server name/cache JNDI name`*. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The Application Server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all the disk cache content at the old location is lost when you restart the Application Server

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if `Enable Disk Offload` is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom performance	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is custom.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	10 to MAXINT

Eviction policy algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The disk cache garbage collector is awaken when the disk size exceeds high threshold limit. The lower value limits disk cache size in GB and disk cache size in entries. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy ends. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value limits disk cache size in GB and disk cache size in entries. The disk cache garbage collector, when awaken, evicts entries until the disk size reaches the low threshold limit. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Use listener context:

Set this value to true to have invalidation events sent to registered invalidation listeners using the Java 2 Platform, Enterprise Edition (J2EE) context of the listener. If you want to use listener J2EE context for callback, set this value to **true**. If you want to use the caller thread context for callback, set this to **false**.

Dependency ID support:

Specifies that the dynamic cache service, supports cache entry dependency IDs. Disable this option if you do not need to use dependency IDs. Dependency IDs specify additional cache group identifiers that associate multiple cache entries to the same group identifier in your cache policy.

This option might not be available for cache instances that were created with a previous version of WebSphere Application Server.

Default	true

Enable cache replication:

Use cache replication to enable sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

This option might be unavailable for cache instances created with a previous version of WebSphere Application Server.

Full group replication domain:

Specifies a replication domain from which your data is replicated.

Specifies a replication domain from which your data is replicated. Choose from any replication domains that have been defined. If there are no replication domains listed, you must create one during cluster creation or manually in the administrative console by clicking **Environment > Internal replication domains > New**. The replication domain you choose to use with the dynamic cache service must be using a Full group replica. Do not share replication domains between replication consumers. Dynamic cache should use a different replication domain from session manager or stateful session beans.

Replication type:

Specifies the global sharing policy for this cache instance.

The following settings are available:

- **Both push and pull** sends the cache ID of newly updated content to other servers in the replication domain. Then, if one of the other servers requests the content, and that server has the ID of the cache entry for the previously updated content, it will retrieve the content from the publishing server. If a request is made for an ID which has not been previously published, the server assumes it does not exist in the cluster and creates a new entry.
- **Pull only** shares cache entries for this object between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
- **Push only** sends the cache ID and cache content of new content to all other servers in the replication domain.
- The sharing policy of **Not Shared** results in the cache ID and cache content not being shared with other servers in the replication domain.

When enabling replication, the default value is **Not Shared**.

Push frequency:

Specifies the time, in seconds, to wait before pushing new or modified cache entries to other servers.

A value of `0` (zero) sends the cache entries immediately. Setting this property to a value greater than `0` (zero) results in a ″batch″ push of all cache entries that are created or modified during the time period. The default is `1` (one).

Object cache instance collection

Use this page to configure and manage object cache instances, which in addition to the default shared dynamic cache, can store, distribute, and share data for Java 2 Platform, Enterprise Edition (J2EE) applications. Use cache instances to give applications better flexibility and tuning of the cache resources.

To view this administrative console page, click **Resources > Cache instances > Object cache instances**.

Use the DistributedObjectCache programming interface to access the cache instances. For more information about the DistributedObjectCache application programming interface, see the API documentation.

Scope: Specify NODE SCOPE to view and configure cache instances that are available to all servers with the particular node. Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Use this name when looking up a reference to this cache instance. The results return a `DistributedMap` object.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands. The default is 2000.

The minimum value is 100, with no set maximum value.

cacheinstances.properties file

Use the information in this document as a reference of the names, values, and explanations that you can use in the cacheinstances.properties file.

The following list provides the property names, associated values, and explanations for the cacheinstance.properties file.

Property name - *x* is the instance number, which starts with `0`	**Version**	**Scope**	**Possible value**	**Description**
Cache core properties				
`cache.instance.x`	5.1.x and later	Per cache instance	any string (no default set)	Specifies cache instance name or JNDI name.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x. cacheSize	5.1.x and later	Per cache instance	> 0 (default=2000)	Specifies the maximum number of entries that are held in memory cache.
cache.instance.x. disableDependencyId	6.0.2.x and later	Per cache instance	True or false (default=false)	Specifies that the dynamic cache service supports cache entry dependency IDs. Disable this option if you do not need to use dependency IDs. Dependency IDs specify additional cache group identifiers that associate multiple cache entries to the same group identifier in your cache policy.
cache.instance.x. disableTemplates Support	6.0.2.x and later	Per cache instance	True or false (default=false)	Specifies whether template support feature is enabled.
cache.instance.x. useListenerContext	5.1.x and later	Per cache instance	True or false (default=false)	Set this value to true to have invalidation events sent to registered invalidation listeners, using the Java Platform, Enterprise Edition (Java EE) context of the listener. If you want to use listener Java EE context for callback, set this value to true. If you want to use the caller thread context for callback, set this value to false.
cache.instance.x. enableNioSupport	6.0.2.x and later	Per cache instance	True or false (default=false)	Specifies whether DistributedMap or DistributedNioMap is used.
cache.instance.x. memoryCacheSizeInMB	7.0	Per cache instance	> 0 (default: -1 limit does not exist)	Specifies a value for the maximum memory cache size in megabytes (MB)

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x.memoryCacheHighThreshold	7.0	Per cache instance	> 0 % (default=95)	Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the memory cache size in MB. The higher value is used when limit memory cache size in MB is specified.
cache.instance.x.memoryCacheLowThreshold	7.0	Per cache instance	> 0 % (default=80)	Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the memory cache size in MB. The lower value is used when limit memory cache size in MB is specified.
cache.instance.x.createCacheAtServerStartup	7.0	Per cache instance	True or false (default=false)	Specifies whether the configured cache instance is created during the server startup. This is useful when cache replication feature is used. However, the time for server startup will take long.
Cache servlet/JavaServer Pages (JSP) caching properties				

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x. cascadeCachespec Properties	6.0.2.19, 6.1.0.9 and later	Per cache instance	True or false (default=false)	A configurable change in the behavior of the cache so that the child pages and fragments inherit the cache specification properties of their parent pages and fragments. If the request for a fragment does not match a defined cache policy, the fragment will inherit the save-attributes and the store-cookies properties from its parent fragment. Enable this cascade of save-attributes and store-cookies properties by setting the value to true.
cache.instance.x. disableStoreCookies	6.0.2.9, 6.1.x and later	Per cache instance	″none″, ″ALL″,″All″, cache instance name, comma delineated list of cookie names, (default=″none″)	Specifies whether disable store cookies is NONE or ALL. Stores cookies as part of the response by default unless configured otherwise on a per request basis in cachespec.xml file. There is a risk of sharing cookies between users, which violates security.
cache.instance.x. enableServlet Support	6.0.2.x and later	Per cache instance	True or false (default=false)	Specifies whether the cache instance is servlet cache or object cache.
Cache disk offload properties				
cache.instance.x. enableDiskOffload	5.1.x and later	Per cache instance	True or false (default=false)	Specifies whether disk offload is enabled.
cache.instance.x. diskOffload Location	5.1.x and later	Per cache instance	String – For example: . $(app_server_root) /diskOffload	Specifies the location on the disk to save cache entries when disk offload is enabled.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x.diskCacheSize	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	>= 0 (0=limit does not exist)	Specifies a value for the maximum disk cache size in number of entries.
cache.instance.x.diskCacheSizeInGB	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	0 or > 2 in GB (0=limit does not exist)	Specifies a value for the maximum disk cache size in gigabytes (GB).
cache.instance.x.diskCacheEntrySizeInMB	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	>= 0 in MB (0=limit does not exist)	Specifies a value for the maximum size of an individual cache entry in megabytes (MB). Any cache entry that is larger than this, when evicted from memory, will not be offloaded to disk.
cache.instance.x.flushToDiskOnStop	5.1.x and later	Per cache instance	True or false (default = false)	Specifies if in-memory cached objects are saved to disk when the server stops.
cache.instance.x.diskCache PerformanceLevel	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	0=low 1=balance 2=custom 3=high (default=1)	Specifies the performance level to tune the performance of the disk cache.
cache.instance.x.htodCleanup Frequency	5.1.1.2 and later	Per cache instance	0 <= x <= 1440 in minutes (0=cleanup at midnight)	Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x.htodDelayOffloadDepIdBuckets	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	> 0 (default=1000)	Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded, the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.
cache.instance.x.htodDelayOffloadTemplateBuckets	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	> 0 (default=100)	Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded, the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.
cache.instance.xhtodDelayOffloadEntriesLimit	5.1.1.2 and later	Per cache instance	> 0 (default=1000)	Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded, the information is offloaded to the disk. This setting applies only when the disk offload performance level is custom.
cache.instance.x.diskCacheEvictionPolicy	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	0=disable 1=random 2:size (default=0)	Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x. diskCacheHigh Threshold	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	> 0 % (default=80)	Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The high value is used when limit disk cache size in GB and limit disk cache size in entries are specified.
cache.instance.x. diskCacheLow Threshold	5.1.1.13, 6.0.2.17, 6.1.x and later	Per cache instance	> 0 % (default=70)	Specifies when the eviction policy runs. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value is used when limit disk cache size in GB and limit disk cache size in entries are specified.
Cache replication properties				
cache.instance.x. enableCache Replication	6.0.2.x and later	Per cache instance	True or false (default=false)	Specifies whether cache replication is enabled. Use cache replication to have cache entries copied to multiple application servers configured in the same replication domain.
cache.instance.x. replicationType	5.1.x and later	Per cache instance	1 (Not shared, 2 (Push), 4 (Push and pull)	Specifies the global sharing policy for this application server.
cache.instance.x. replicationDomain	6.0.2.x and later	Per cache instance	String – For example: DynamicCacheDomain	Specifies a replication domain from which your data is replicated.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x. useServer ClassLoader	5.1.1.9, 6.0.2.9, 6.1.x and later	Per cache instance	True or false (default=false)	Specifies whether using server class loader is enabled. Setting this value to true, deserializes the InvalidationEvent using system classloader first and then using application classloader, if that fails. This improves performance.
cache.instance.x. cacheEntryWindow	5.1.1.13, 6.0.2.17, 6.1.0.7 and later	Per cache instance	> 0 (default=50)	Specifies a limit on the total number of cache entries that are sent by the data replication service (DRS) in terms of number of entries.
cache.instance.x. cachePercentage Window	5.1.1.13, 6.0.2.17, 6.1.0.7 and later	Per cache instance	> 0 % (default=2)	Specifies a limit on the number of cache entries that are sent by DRS in terms of the percentage of total cache in memory.
cache.instance.x. cacheInvalidate EntryWindow	5.1.1.14, 6.0.2.19, 6.1.0.7 and later	Per cache instance	> 0 (default=50)	Specifies a limit on the total number of invalidation events that are sent by DRS in terms of number of entries.
cache.instance.x. cacheInvalidate PercentWindow	5.1.1.14, 6.0.2.19, 6.1.0.7 and later	Per cache instance	> 0 % (default=2)	Specifies a limit on the number of invalidation events that are sent by DRS in terms of the percentage of total cache in memory.
cache.instance.x. filterTimeOut Invalidation	6.0.2.13, 6.1.x and later	Per cache instance	True or false (default=false)	Specifies whether sending invalidations that are based on timeout eviction is enabled.
cache.instance.x. filterLRU Invalidation	6.0.2.13, 6.1.x and later	Per cache instance	True or false (default=false)	Specifies whether sending invalidations that are based on LRU eviction is enabled.

Property name - *x* is the instance number, which starts with 0	Version	Scope	Possible value	Description
cache.instance.x. ignoreValueIn InvalidationEvent	5.1.1.13, 6.0.2.17, 6.1.x or later	Per cache instance	True or false (default=false)	Specifies whether the cache value of Invalidation event is ignored. If it is true, the cache value of Invalidation event is set to NULL when the code is returned to the caller.

Invalidation listeners

Invalidation listener mechanism uses Java events for alerting applications when contents are removed from the cache.

Applications implement the InvalidationListener interface (defined in the com.ibm.websphere.cache package) and register it to the cache using the DistributedMap interface. Listeners receive InvalidationEvents (defined in the com.ibm.websphere.cache package) when entries from the cache are removed, due to an explicit user invalidation, timeout, least recently used (LRU) eviction, cache clear, or disk timeout. Applications can immediately recalculate the invalidated data and prime the cache before the next user request.

Enable listener support in DistributedMap before registering listeners. DistributedMap can also be configured to use the invalidation listener Java Platform, Enterprise Edition (Java EE) context from registration time during callbacks. Setting the value of the custom property useListenerContext to true enables the invalidation listener Java EE context for callbacks. See Cache instance settings for more information.

The following example shows how to set up an invalidation listener:

```
dmap.enableListener(true);  // Enable cache invalidation listener.
InvalidationListener listener = new MyListenerImpl();  //Create invalidation listener object.
dmap.addInvalidationListener(listener);  //Add invalidation listener.
      :
      :
      :
dmap.removeInvalidationListener(listener);  //Remove the invalidation listener.
//This increases performance.
dmap.enableListener(false);  // Disable cache invalidation listener.
//This increases performance.
```

For more information about invalidation listeners, see Additional Application Programming Interfaces (APIs) for the com.ibm.websphere.cache package.

Disabling template-based invalidations during JSP reloads

By setting the Java virtual machine (JVM) com.ibm.ws.cache.CacheConfig.disableTemplateInvalidation custom property to **true**, the template-based invalidations are disabled during JSP reloads.

About this task

To set any of these JVM custom properties, complete the following steps:

1. In the administrative console, click **Servers** > **Application servers** > ***server_name*** > **Process definition** > **Java virtual machine** > **Custom properties** > **New**.
2. Enter com.ibm.ws.cache.CacheConfig.disableTemplateInvalidation in the **Name** field.

3. Enter `true` in the **Value** field.
4. Save the property and restart WebSphere Application Server.

Using object cache instances

Perform this task so that your application can access dynamic cache object cache instances with the DistributedMap or DistributedObjectCache interfaces.

Before you begin

Before you begin, enable the dynamic cache service. See “Using the dynamic cache service” on page 2210 for more information.

About this task

Using object cache instances can improve the performance of your application because you can programmatically store and share frequently used objects. By using object cache instances, you also have the necessary control over the dynamic cache when you are running multiple applications in an application server. See “Cache instances” on page 1336 for more information.

1. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Object cache instances**.
   b. Specify the scope for the cache instance.

      Node scope makes the cache instance available to all servers on the particular node. Server scope makes the cache instance available to only the selected server. You can mix scopes, if necessary.
   c. Click **Apply** after changing the scope.
   d. Click **New**.
   e. Enter the Java Naming and Directory Interface (JNDI) name for this cache instance.

      This is name that you pass to the InitialContext lookup() method from within your application. For example, `services/cache/instance_one`.
   f. Enter or modify other properties as needed.
2. Update your application. To store and retrieve objects in an object cache instance, you need a DistributedMap or DistributedObjectCache reference for the named object cache instance. See “Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache” on page 2253 for more information.

Results

You configured object cache instances that you can access programmatically with the DistributedMap and DistributedObjectCache interfaces.

Displaying cache information

Use this task to monitor the activity of the dynamic cache service.

About this task

The dynamic cache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information for servlet cache instances.

1. Use the administrative console to install the cache monitor application from the *app_server_root*`/installableApps` directory. The name of the application is `CacheMonitor.ear`. For more information about installing applications, see Installing enterprise application files with the console. Install the cache monitor onto the application server that you want to monitor.
2. Configure the Web container transport chain and host alias for the server with cache monitor installed.

a. Add a host alias for the port your server is using. Click **Environment > Virtual hosts >** *host_type* **> Host aliases** and create a new **Host name** and **Port** to add to the list.
b. You can then access the cache monitor using `http://your_host_name:your_port_number/cachemonitor`.

Note: You can find the port number in the `SystemOut.log` file. Look for message `TCPC0001I` or `SRVE0171I`.

3. Access the cache monitor using a Web browser and the URL `http://your host_name:your port_number/cachemonitor`, where *your port_number* is the port associated with the host on which you installed the cache monitor application.
4. Verify the list of cache instances that are shown. For each cache instance, you can perform the following actions:

   Note: You must select the servlet cache instance that you want to monitor. If you do not use servlet cache instances by using <cache-instance> tags in your cachespec.xml file, all the content is in the **baseCache** instance.

   - View the Statistics page and verify the cache configuration and cache data. Click **Reset Statistics** to reset the counters.
   - View the Cache Policies page to see which cache policies are currently loaded in the dynamic cache. Click on a template to view the cache ID rules for the template.
   - View the Cache Contents page to examine the contents that are currently cached in memory.
   - View the Edge Statistics page to view data about the current ESI processors configured for caching. Click **Refresh Statistics** to see the latest statistics or content from the ESI processors. Click **Reset Statistics** to reset the counters.
   - View the Disk Offload page to view the disk configuration, statistics, and the content that is currently off-loaded from memory to disk.

   When you are viewing contents on memory or disk, click on a template to view all entries for that template, click on a dependency ID to view all entries for the ID, or click on the cache ID to view all the data that is cached for that entry.
5. Use the cache monitor to perform basic operations on data in a cache instance.

   Remove an entry from cache
   Click **Invalidate** when viewing a cache entry.
   Remove all entries for a certain dependency ID
   Click **Invalidate** when viewing entries for a dependency ID.
   Remove all entries for a certain template
   Click **Invalidate** when viewing entries for a template.
   Move an entry to the front of the Least Recently Used queue to avoid eviction
   Click **Refresh** when viewing a cache entry.
   Move an entry from disk to cache
   Click **Send to Memory** when viewing a cache entry on disk.
   Clear the entire contents of the cache
   Click **Clear Cache** while viewing statistics or contents.
   Clear the contents on the ESI processors
   Click **Clear Cache** while viewing ESI statistics or contents.
   Clear the contents of the disk cache
   Click **Clear Disk** while viewing disk contents.

Cache monitor

Cache monitor is an installable Web application that provides a real-time view of the current state of dynamic cache. You use it to help verify that dynamic cache is operating as expected. The only way to manipulate the data in the cache is by using the cache monitor. It provides a GUI interface to manually change data.

Cache monitor provides a way to:

- **Verify the configuration of dynamic cache**

  After you create multiple servlet cache instances in the administrative console, you can configure properties, including the maximum size of the cache and disk offload location on each cache instance, as well as advanced features such as controlling external caches. You can verify the configuration of the dynamic cache by viewing of the configured features and properties in the cache monitor.
- **Verify the cache policies**

  To cache an object, unique IDs must be generated for different invocations of that object. To create unique IDs for each object, provide rules for each cacheable object in the `cachespec.xml` file, found inside the Web module `WEB-INF` or `enterprise bean` `META-INF` directory. See “cachespec.xml file” on page 2233 for more information. Each cacheable object can have multiple cache ID rules that run in sequence until either a rule returns a cache ID or no more rules remain. If none of the cache ID generation rules produce a valid cache ID, then the object is not cached. There can be multiple `cachespec.xml` files with multiple cache ID rules. With cache monitor, you can verify the policies of each object. You can also view all of the cache polices for each cache instance that is currently loaded in dynamic cache. This view is also convenient to verify that the `cachespec.xml` file was read by the dynamic cache without errors.
- **Monitor cache statistics**

  You can view the essential cache data, such as number of cache hits, cache misses, and number of entries in each cache instance. With this data, you can tune the cache configuration to improve the dynamic cache performance. For example, if the number of used entries is often high, and entries are being removed and recreated, consider increasing the maximum size of the cache or enabling disk offload.
- **Monitor the data flowing through the cache**

  Once a cacheable object is invoked, dynamic cache creates a cache entry for it that contains the output of the actions that are performed and metadata, such as time to live, sharing policy, and so on. Entries are distinguished by a unique ID string that is based on the rules specified in the `cachespec.xml` file for the particular object name. Objects with the same name might generate multiple cache IDs for different invocations, based on request parameters and attributes for each invocation. You can view of all the cache entries that are in the cache instance, based on the unique ID. You can also view the group of cache entries that share a common name (also known as template). Cache entries can also be grouped together by a dependency ID, which is used to invalidate the entire group of entries dependent on a common entity. Therefore, cache monitor also provides a view of the group of cache entries that share a common dependency ID.

  For each entry, cache monitor also displays metadata, such as time to live, priority and sharing-policy, and provides a view of the output that has been cached. This helps the customer to verify which pages have been cached, that the pages have been cached in the correct cache instance with the right attributes such as time to live, priority, and that the pages have the right content.
- **Monitor the data in the edge cache**

  Dynamic cache provides support to recognize the presence of an Edge Side Include (ESI) processor and to generate ESI include tags and appropriate cache policies for edge cacheable fragments. With the ESI processor, you can cache whole pages, as well as fragments, providing a higher cache hit ratio. There can be multiple ESI processors running on multiple hosts configured for caching.

  You can view a list of all ESI processes and their hosts that are enabled for caching. Select a host or a processor, and view its edge cache statistics and the current cache entries.
- **View the data offloaded to the disk**

  By default, when the number of cache entries reaches the configured limit for a given server, cache entries are removed, enabling new entries to enter the cache service. With disk offload, the removed cache entries are copied to disk for future access. You can view the content that is copied to disk that corresponds to the view of the contents cached in memory for each cache instance.
- **Manage the data in the cache**

  You can perform the following basic operations on the data in the cache:
  - Remove an entry from a cache instance
  - Remove all entries for a certain dependency ID

- Remove all entries for a certain name (template)
- Move an entry to the front of the least recently used queue to avoid removal of the cache entry
- Move an entry from the disk to the memory within a cache instance
- Clear the entire contents of the cache instance
- Clear the contents of the disk for the cache instance

With these operations, you can manually change the state of the cache without having to restart the server.

Edge cache statistics:

Cache monitor provides a view of the edge cache statistics.

The following statistics are available:

- **ESI Processors**. The number of processes that are configured as edge caches.
- **Number of Edge Cached Entries**. The number of entries that are currently cached on all edge servers and processes.
- **Cache Hits**. The number of requests that match entries on edge servers.
- **Cache Misses By URL.** A cache policy does not exist on the edge server for the requested template.

  Note:
  - The initial ESI request for a template that has a cache policy on WebSphere Application Server results in a miss.
  - Every request for a template that does not have a cache policy on WebSphere Application Server results in a miss by URL on the edge server.
- **Cache Misses By Cache ID**. The policy for the requested template exists on the edge server, and a cache ID is created, based on the ID rules and the request attributes, but the cache entry for this ID does not exist.

  Note: If the policy exists on the edge server for the requested template, but a cache ID match is not found, based on the ID rules and the request attributes, it is not treated as a cache miss.
- **Cache Timeouts**. The number of entries that are removed from the edge cache based on the timeout value.
- **Evictions**. The number of entries that are removed from the edge cache due to invalidations received from WebSphere Application Server.

Tuning dynamic cache with the cache monitor

Use this task to interpret cache monitor statistics to improve the performance of the dynamic cache service.

Before you begin

Verify that dynamic cache is enabled and that the cache monitor application is installed on your application server.

About this task

See the Displaying cache information topic in the *Administering applications and their environment* PDF to configure the cache monitor application.

Use the cache monitor to watch cache hits versus misses. By comparing these two values, you can determine how much dynamic cache is helping your application, and if you can take any additional steps to further improve performance and decrease the cost of processing for your application server.

1. Start cache monitor and click on **Cache Statistics**. You can view the following cache statistics:

Cache statistic	Description
Cache Size	The maximum number of entries that the cache can hold.
Used Entries	The number of cache entries used.
Cache Hits	The number of request responses that are served from the cache.
Cache Misses	The number of request responses that are cacheable but cannot be served from the cache.
LRU Evictions	The number of cache entries removed to make room for new cache entries.
Explicit Removals	The number of cache entries removed or invalidated from the cache based on cache policies or were deleted from the cache through the cache monitor.

2. You can also view the following cache configuration values:

Cache configuration value	Description
Default priority	Specifies the default priority for all cache entries. Lower priority entries are moved from the cache before higher priority entries when the cache is full. You can specify the priority for individual cache entries in the cache policy.
Servlet Caching Enabled	If servlet caching is enabled, results from servlets and JavaServer Pages (JSP) files are cached. See the *Administering applications and their environment* PDF for more information.
Disk Offload Enabled	Specifies if entries that are being removed from the cache are saved to disk. See the *Administering applications and their environment* PDF for more information.

3. Wait for the application server to add data to the cache. You want the number of used cache entries in the cache monitor to be as high as it can go. When the number of used entries is at its highest, the cache can serve responses to as many requests as possible.
4. When the cache has a high number of used entries, reset the statistics. Watch the number of cache hits versus cache misses. If the number of hits is far greater than the number of misses, your cache configuration is optimal. You do not need to take any further actions. If you find a higher number of misses with a lower number of hits, the application server is working hard to generate responses instead of serving the request using a cached value. The application server might be making database queries, or running logic to respond to the requests.
5. If you have a large number of cache misses, increase the number of cache hits by improving the probability that a request can be served from the cache.

   To improve the number of cache hits, you can increase the cache size or configure additional cache policies. See the *Administering applications and their environment* PDF for more information to increase the cache size and to configure cache policies.

Results

By using the cache monitor application, you optimized the performance of the dynamic cache service.

What to do next

See the *Administering applications and their environment* PDF for more information about the dynamic cache.

Using servlet cache instances

Use this task to configure servlet cache instances.

Before you begin

Before you begin, enable the dynamic cache service. See “Using the dynamic cache service” on page 2210 for more information.

About this task

Perform this task so that your application can access dynamic cache servlet cache instances. Using servlet cache instances can improve the performance of your application because you can store the output and the side effects of an invoked servlet. Servlet cache instances also give you the necessary control over the cache for multiple applications that are running in an application server. See “Cache instances” on page 1336 for more information.

1. Enable servlet caching. See “Configuring servlet caching” on page 2215 for more information.
2. Configure one or more cache instances.
   a. In the administrative console, click **Resources > Cache instances > Servlet cache instances**.
   b. Specify the scope of the cache instance. Specify a scope of cell to make the cache instance available to all the servers that are in the cell. Node scope makes the cache instance available to all servers in a node. Server scope makes the cache instance available to the selected server only. If necessary, you can mix the scopes.
   c. Click **Apply** to save the scope.
   d. Specify the settings for the cache instance. The **Name** and **Java Naming and Directory interface (JNDI)** name fields are required. The **JNDI name** is the name attribute that is specified in the <cache-instance> element in the `cachepec.xml` file. An example of a JNDI name that is specified in the cachespec.xml file follows:

      ```
      <cache-instance name="services/cache/instance_one">
      ```

      In this example, specify `services/cache/instance_one` as the **JNDI name**.
3. Update your application. To use a servlet cache instance, you must specify a <cache-instance> element that has a name that is equal to the JNDI Name for this cache instance.

Servlet cache instance collection

A servlet cache instance is a location, in addition to the default shared dynamic cache, where dynamic cache can store, distribute, and share data. By using servlet cache instances, your applications have greater flexibility and better tuning of the cache resources. The Java Naming and Directory Interface (JNDI) name specified for the cache instance is mapped to the name attribute in the <cache-instance> tag in the `cachespec.xml` configuration file.

To view this administrative console page, click **Resources > Cache instances > Servlet cache instances**.

Scope: Specify NODE SCOPE to view and configure cache instances that are available to all servers with the particular node. Specify SERVER SCOPE to view and configure cache instances that are available only on the specific server.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Specify this name in the name attribute field in the <cache-instance> tag in the `cachespec.xml` configuration file. This tag is used to find the particular cache instance in which to store cache entries.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands. The default is 2000.

The minimum value is 100, with no set maximum value.

Servlet cache instance settings

A servlet cache instance is a location, in addition to the default shared dynamic cache, where dynamic cache can store, distribute, and share data. By using servlet cache instances, your applications have greater flexibility and better tuning of the cache resources. The Java Naming and Directory Interface (JNDI) name specified for the cache instance is mapped to the name attribute in the <cache-instance> tag in the `cachespec.xml` configuration file.

To view this administrative console page, click **Resources > Cache instances > Servlet cache instances > *cache_instance_name***.

Name:

Specifies the required display name for the resource.

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the resource. Specify this name in the name attribute field in the <cache-instance> tag in the `cachespec.xml` configuration file. This tag is used to find the particular cache instance in which to store cache entries.

Description:

Specifies a description for the resource. This field is optional.

Category:

Specifies a category string to classify or group the resource. This field is optional.

Cache size:

Specifies a positive integer for the maximum number of entries the cache holds. The cache size is usually in the thousands.

Default	2000
Range	100 - no set maximum value

Default priority:

Specifies the default priority for servlets that can be cached. This value determines how long an entry stays in a full cache.

The recommended value is one.

Enable disk offload:

Specifies if disk offloading is enabled.

If you have disk offload disabled, when a new entry is created while the cache is full, the priorities are configured for each entry and the least recently used algorithm are used to remove the entry from the cache in memory. If you enable disk offload, the entry that would be removed from the cache is copied to the local file system. The location of the file is specified by the disk offload location.

Default	false

Offload location:

Specifies the directory used for disk offload.

If disk offload location is not specified, the default location, `${WAS_TEMP_DIR}/node/server name/_dynacache/cache JNDI name` will be used. If disk offload location is specified, the node, server name, and cache instance name are appended. For example, `${USER_INSTALL_ROOT}/diskoffload` generates the location as `${USER_INSTALL_ROOT}/diskoffload/node/server name/cache JNDI name`. This value is ignored if disk offload is not enabled.

The default value of the ${WAS_TEMP_DIR} property is `${USER_INSTALL_ROOT}/temp`. If you change the value of the ${WAS_TEMP_DIR} property after starting WebSphere Application Server, but do not move the disk cache contents to the new location:

- The Application Server creates a new disk cache file at the new disk offload location.
- If the Flush to disk setting is enabled, all the disk cache content at the old location is lost when you restart the Application Server

Flush to disk:

Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if `Enable Disk Offload` is not selected.

Default	false

Limit disk cache size in GB:

Specifies a value for the maximum disk cache size in GB. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache size in entries:

Specifies a value for the maximum disk cache size in number of entries. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Limit disk cache entry size:

Specifies a value for the maximum size of an individual cache entry in MB. Any cache entry larger than this, when evicted from memory, will not be offloaded to disk. When you select this option, you can specify a positive integer value. Leaving this option blank indicates an unlimited size. This setting applies only if enable disk offload is specified for the cache.

Value	0 to MAXINT. A value of 0 indicates unlimited size.

Performance settings:

Specifies the level of performance that is required by the disk cache. This setting applies only if **enableDiskOffload** is specified for the cache. Performance levels determine how memory resources should be used on background activity such as cache cleanup, expiration, garbage collection, and so on. This setting applies only if enable disk offload is specified for the cache.

High performance and high memory usage	Indicates that all metadata will be kept in memory.
Balanced performance and balanced memory usage	Indicates some metadata will be kept in memory. This is the default performance setting and will provide an optimal balance of performance and memory usage for most users.
Low performance and low memory usage	Indicates that limited metadata will be kept in memory.
Custom	Indicates that the administrator will explicitly configure the memory settings that will be used to support the above background activity. The administrator sets these values using the **DiskCacheCustomPerformanceSettings** object.

Disk cache cleanup frequency:

Specifies a value for the disk cache cleanup frequency, in minutes. If this value is set to 0, the cleanup runs only at midnight. This setting applies only when the Disk Offload Performance Level is low, balanced, or custom. The high performance level does not require disk cleanup, and this value is ignored.

Value	0 to 1440

Maximum buffer for cache identifiers per metaentry:

Specifies a value for the maximum number of cache identifiers that are stored for an individual dependency ID or template in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk offload performance level is CUSTOM.

Value	100 to MAXINT

Maximum buffer for dependency identifiers:

Specifies a value for the maximum number of dependency identifier buckets in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is custom.

Value	100 to MAXINT

Maximum buffer for templates:

Specifies a value for the maximum number of template buckets that are in the disk cache metadata in memory. If this limit is exceeded the information is offloaded to the disk. This setting applies only when the disk cache performance level is cusotm.

Value	10 to MAXINT

Eviction policy algorithm:

Specifies the eviction algorithm that the disk cache will use to evict entries once the high threshold is reached. This setting applies only if enable disk offload is specified for the cache.

None	No eviction policy, so the disk cache can grow until it reaches its limit at which time the dynamic cache service stops writing to disk
Random	When the disk size reaches a high threshold limit, the disk cache garbage collector wakes up and randomly picks entries on the disk and evicts them until the size reaches a low threshold limit.
Size	When the disk size reaches a high threshold limit, the disk cache garbge collector wakes up and picks the largest entries on the disk and evicts them until the disk size reaches a low threshold limit.

High threshold:

Specifies when the eviction policy starts. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The disk cache garbage collector is awaken when the disk size exceeds high threshold limit. The lower value limits disk cache size in GB and disk cache size in entries. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Low threshold:

Specifies when the eviction policy ends. The threshold is expressed in terms of the percentage of the disk cache size in GB or entries. The lower value limits disk cache size in GB and disk cache size in entries. The disk cache garbage collector, when awaken, evicts entries until the disk size reaches the low threshold limit. This setting does not apply when the disk cache eviction policy is set to none.

Values	1 to 100

Enable cache replication:

Use cache replication to enable sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

This option might be unavailable for cache instances created with a previous version of WebSphere Application Server.

Full group replication domain:

Specifies a replication domain from which your data is replicated.

Specifies a replication domain from which your data is replicated. Choose from any replication domains that have been defined. If there are no replication domains listed, you must create one during cluster creation or manually in the administrative console by clicking **Environment > Internal replication domains > New**. The replication domain you choose to use with the dynamic cache service must be using a Full group replica. Do not share replication domains between replication consumers. Dynamic cache should use a different replication domain from session manager or stateful session beans.

Replication type:

Specifies the global sharing policy for this cache instance.

The following settings are available:

- **Both push and pull** sends the cache ID of newly updated content to other servers in the replication domain. Then, if one of the other servers requests the content, and that server has the ID of the cache entry for the previously updated content, it will retrieve the content from the publishing server. If a request is made for an ID which has not been previously published, the server assumes it does not exist in the cluster and creates a new entry.
- **Pull only** shares cache entries for this object between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server runs the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended.
- **Push only** sends the cache ID and cache content of new content to all other servers in the replication domain.
- The sharing policy of **Not Shared** results in the cache ID and cache content not being shared with other servers in the replication domain.

When enabling replication, the default value is **Not Shared**.

Push frequency:

Specifies the time, in seconds, to wait before pushing new or modified cache entries to other servers.

A value of `0` (zero) sends the cache entries immediately. Setting this property to a value greater than `0` (zero) results in a ″batch″ push of all cache entries that are created or modified during the time period. The default is `1` (one).

Using the DynamicContentProvider interface for dynamic cache

Use this task to configure the DynamicContentProvider interface for cached servlets and JavaServer Pages (JSP) files.

Before you begin

The dynamic cache service should be enabled and you should be using servlet caching. See “Using the dynamic cache service” on page 2210 and “Configuring servlet caching” on page 2215 for more information.

About this task

A cacheable servlet or JavaServer Pages (JSP) file might contain a state in the response that does not belong to the fragment for that servlet or JSP. When the state changes, the cached servlet or JSP is not valid for caching. Use the com.ibm.websphere.servlet.cache.DynamicContentProvider interface to make the fragment cacheable.

Servlets or JSP files that implement the DynamicContentProvider interface can add user exits in fragments that are cacheable by calling the addDynamicContentProvider(DCP) method on the wrapper response

object. When the dynamic cache renders the page, it identifies the user exit and calls the dynamic content provider to add the dynamic content to the rendered page.

1. Provide an implementation class of the com.ibm.websphere.servlet.cache.DynamicContentProvider interface. An example of an implementation follows:

```
class DynamicContentProviderImpl implements com.ibm.websphere.servlet.cache.
   DynamicContentProvider {
    DynamicContentProviderImpl() {}

    public void provideDynamicContent(HttpServletRequest request, OutputStream streamWriter)
           throws IOException {
     String dynamicContent = System.currentTimeMillis();
     streamWriter.write(dynamicContent.getBytes());
    }
    public void provideDynamicContent(HttpServletRequest request, Writer streamWriter)
           throws IOException {
     String dynamicContent = System.currentTimeMillis();
     streamWriter.write(dynamicContent.toCharArray());
    }
   }
```

2. Add user exits to your servlet or JSP file by calling the addDynamicContentProvider(DCP) method on the wrapper response object. An example follows:

```
public class DCPServlet extends CacheTestCase {
    public void performTest(HttpServletRequest request, HttpServletResponse response)
           throws IOException, ServletException {
     out.println("Testing the DCP feature begin "+System.currentTimeMillis());
     DynamicContentProvider dcp = new DynamicContentProviderImpl();
     ServletCacheResponse scr = (ServletCacheResponse)(response);
     scr.addDynamicContentProvider(dcp);
       out.println("Testing the DCP feature end"+System.currentTimeMillis());
    }
   }
```

What to do next

See “Task overview: Using the dynamic cache service to improve performance” on page 2197 for more information about the other tasks that you can perform with the dynamic cache.

Dynamic query

Using EJB query

The Enterprise JavaBeans (EJB) query language is used to specify a query over container-managed entity beans. The language is similar to structured query language (SQL). An EJB query is independent of the bean’s mapping to a persistent store.

About this task

An EJB query can be used in three situations:

- To define a finder method of an EJB entity bean.
- To define a select method of an EJB entity bean.
- To dynamically specify a query using the executeQuery method dynamic API.

Finder and select queries are specified in the bean’s deployment descriptor using the <ejb-ql> tag; they are compiled into SQL during deployment. Dynamic queries are included within the application code itself.

The product’s EJB query language is compliant with the EJB QL defined in Sun’s EJB 2.1 and EJB 3.0 specifications and has additional capabilities as listed in the topic Comparison of EJB specification and WebSphere Query Language.

- Before using EJB query, familiarize yourself with query language concepts, starting with the topic, EJB Query Language.
- Define an EJB query in one of the following ways:
  - **Rational Application Developer.** When defining an entity bean, specify the `<ejb-ql>` tag for the finder or select method. For more information about using Rational Application Developer see the assembly tool information center at http://wilson.boulder.ibm.com/infocenter/radhelp/v7r5mbeta/index.jsp?topic=/com.ibm.jee5.doc/topics/cejb3.html
  - **Dynamic query service.** Add the executeQuery method to your application.

Example

See the topic Example: EJB queries.

EJB query language

EJB query language enables you to write queries based on entity beans without knowing the underlying relational schema.

An EJB query is a string that contains the following elements:
- a SELECT clause that specifies the enterprise beans or values to return;
- a FROM clause that names the bean collections;
- an optional WHERE clause that contains search predicates over the collections;
- an optional GROUP BY and HAVING clause (see Aggregation functions);
- an optional ORDER BY clause that specifies the ordering of the result collection.

Collections of entity beans are identified in EJB queries through the use of their abstract schema name in the query FROM clause.

The elements of EJB query language are discussed in more detail in the following related topics.

Example: Queries with EJB:

Here is an example Enterprise JavaBeans (EJB) schema, followed by a set of example queries.

Table 53. DeptBean schema

Entity bean name (EJB name)	DeptEJB (not used in query)
Abstract schema name	DeptBean
Implementation class	com.acme.hr.deptBean (not used in query)
Persistent attributes (cmp fields)	• deptno - Integer (key) • name - String • budget - BigDecimal
Relationships	• emps - 1:Many with EmpEJB • mgr - Many:1 with EmpEJB

Table 54. EmpBean schema

Entity bean name (EJB name)	EmpEJB (not used in query)
Abstract schema name	EmpBean
Implementation class	com.acme.hr.empBean (not used in query)

Table 54. EmpBean schema (continued)

Persistent attributes (cmp fields)	• empid - Integer (key) • name - String • salary - BigDecimal • bonus - BigDecimal • hireDate - java.sql.Date • birthDate - java.util.Calendar • address - com.acme.hr.Address
Relationships	• dept - Many:1 with DeptEJB • manages - 1:Many with DeptEJB

Address is a serializable object used as cmp field in EmpBean. The definition of address is as follows:

```
   public class com.acme.hr.Address  extends Object implements Serializable {
public String street;
public String state;
public String city;
public Integer zip;
     public double distance (String start_location) { ... } ;
     public  String format ( ) { ... } ;
}
```

The following query returns all departments:

```
SELECT OBJECT(d) FROM DeptBean d
```

The following query returns departments whose name begins with the letters ″Web″. Sort the result by name:

```
SELECT OBJECT(d) FROM DeptBean d WHERE  d.name LIKE  'Web%' ORDER BY d.name
```

The keywords SELECT and FROM are shown in uppercase in the examples but are case insensitive. If a name used in a query is a reserved word, the name must be enclosed in double quotes to be used in the query. You can find a list of reserved words in “EJB query: Reserved words” on page 2307. Identifiers enclosed in double quotes are case sensitive. This example shows how to use a cmp field that is a reserved word:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d."select" > 5
```

The following query returns all employees who are managed by Bob. This example shows how to navigate relationships using a path expression:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name='Bob'
```

A query can contain a parameter which refers to the corresponding value of the finder or select method. Query parameters are numbered starting with 1:

```
SELECT OBJECT (e) FROM EmpBean e WHERE e.dept.mgr.name= ?1
```

This query shows navigation of a multivalued relationship and returns all departments that have an employee that earns at least 50000 but not more than 90000:

```
SELECT OBJECT(d) FROM DeptBean d,  IN (d.emps) AS e
WHERE e.salary BETWEEN 50000 and 90000
```

There is a join operation implied in this query between each department object and its related collection of employees. If a department has no employees, the department does not appear in the result. If a department has more than one employee that earns more than 50000, that department appears multiple times in the result.

The following query eliminates the duplicate departments:

```
SELECT DISTINCT OBJECT(d) from DeptBean d,  IN (d.emps) AS e  WHERE e.salary > 50000
```

Find employees whose bonus is more than 40% of their salary:

```
SELECT OBJECT(e) FROM EmpBean e where e.bonus > 0.40 * e.salary
```

Find departments where the sum of salary and bonus of employees in the department exceeds the department budget:

```
SELECT OBJECT(d) FROM DeptBean d where d.budget <
( SELECT SUM(e.salary+e.bonus) FROM IN(d.emps) AS e )
```

A query can contain DB2 style date-time arithmetic expressions if you use java.sql.* datatypes as CMP fields and your datastore is DB2. Find all employees who have worked at least 20 years as of January 1st, 2000:

```
SELECT OBJECT(e) FROM EmpBean e where year(  '2000-01-01' - e.hireDate ) >= 20
```

If the datastore is not DB2 or if you prefer to use java.util.Calendar as the CMP field, then you can use the java millsecond value in queries. The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Find departments with no employees:

```
SELECT OBJECT(d) from DeptBean d where d.emps IS EMPTY
```

Find all employees whose earn more than Bob:

```
SELECT OBJECT(e) FROM EmpBean e, EmpBean b
WHERE b.name = 'Bob' AND e.salary + e.bonus > b.salary + b.bonus
```

Find the employee with the largest bonus:

```
SELECT OBJECT(e) from EmpBean e  WHERE e.bonus =
(SELECT MAX(e1.bonus) from EmpBean e1)
```

The above queries all return EJB objects. A finder method query must always return an EJB Object for the home. A select method query can in addition return CMP fields or other EJB Objects not belonging to the home.

The following would be valid select method queries for EmpBean. Return the manager for each department:

```
SELECT  d.mgr FROM DeptBean d
```

Return department 42 manager's name:

```
SELECT  d.mgr.name FROM DeptBean d WHERE  d.deptno = 42
```

Return the names of employees in department 42:

```
SELECT e.name FROM EmpBean e WHERE  e.dept.deptno=42
```

Another way to write the same query is:

```
SELECT e.name from DeptBean d, IN (d.emps) AS e WHERE d.deptno=42
```

Finder and select queries allow only a single CMP field or EJBObject in the SELECT clause. A select query can return aggregate values in Enterprise JavaBeans 2.1 using SUM, MIN, MAX, AVG and COUNT.

```
SELECT max(e.salary) FROM EmpBean e WHERE e.dept.deptno=42
```

The dynamic query API allows multiple expressions in the SELECT clause. The following query would be a valid dynamic query, but not a valid select or finder query:

```
SELECT  e.name, e.salary+e.bonus as total_pay , object(e), e.dept.mgr
FROM  EmpBean e
ORDER BY 2
```

The following dynamic query returns the number of employees in each department:

```
SELECT e.dept.deptno as department_number , count(*) as employee_count
FROM  EmpBean e
GROUP BY  by e.dept.deptno
ORDER BY 1
```

The dynamic query API allows queries that contain bean or value object methods:

```
SELECT object(e), e.address.format( )
FROM EmpBean e EmpBean e
```

FROM clause:

The FROM clause specifies the collections of objects to which the query is to be applied. Each collection is specified either by an abstract schema name (ASN) or by a path expression identifying a relationship. An identification variable is defined for each collection.

Conceptually, the semantics of the query is to form a temporary collection of tuples, **R**, with elements consisting of all possible combinations of objects from the collections. This collection is subject to the constraints imposed by any path relationships and by the JOIN operation. The JOIN can be either an *inner* or *outer* join.

The identification variables are bound to elements of the tuple. After forming the temporary collection, the search conditions of the WHERE clause are applied to R, and yield a new temporary collection, **R1**. The ORDER BY, GROUP BY, HAVING, and SELECT clauses are applied to R1 to yield the final result.

```
from_clause::=FROM identification_variable_declaration [, {identification_variable_declaration |
                          collection_member_declaration } ]*

identification_variable_declaration::= range_variable_declaration [join]*

join :=  [ { LEFT [OUTER] | INNER  }]  JOIN  {collection_valued_path_expression | single_valued_path_expression}
 [AS] identifier
```

Examples: Joining collections

`DeptBean` contains records 10, 20, and 30. `EmpBean` contains records 1, 2, and 3 that are related to department 10, and records 4 and 5 that are related to department 20. Department 30 has no employees.

```
SELECT d FROM  DeptBean AS d, EmpBean AS e
WHERE d.name = e.name
```

The comma syntax performs an inner join resulting in all possible combinations. In this example, R would consist of 15 tuples (3 departments x 5 employees). If any collection is empty, then R is also empty. The keyword *AS* is optional.

This example shows that a collection can be joined with itself.

```
SELECT d FROM DeptBean AS d, DeptBean AS d1
```

R would consist of 9 tuples (3 departments x 3 departments).

Examples: Relationship joins

A collection can be a relationship based on a previously declared identifier as in

```
SELECT e FROM DeptBean AS d , IN (d.emps) AS e
```

R would contain 5 tuples. Department 30 would not appear in R because it contains no employees. Department 10 would appear in 3 tuples and department 20 would appear in 2 tuples. IN can only refer to multi-valued relationships. The following is not valid

```
SELECT m FROM EmpBean e,  IN( e.dept.mgr) as m   INVALID
```

When joining with a relationship the alternate syntax INNER JOIN (keyword INNER is optional) can also be used, as shown here.

```
SELECT e FROM DeptBean AS d INNER JOIN d.emps AS e
```

An ASN declaration (**d** in the above query) can be followed by one or more join clauses. The relationship following the JOIN keyword must be related (directly or indirectly) to the ASN declaration. Unlike the case with the IN clause, relationships used in a join clause can be single- or multi-valued. This query has the same semantics as the query

```
SELECT e FROM DeptBean AS d , IN (d.emps) AS e
```

You can use multiple joins together.

```
SELECT m FROM EmpBean e JOIN e.dept d JOIN d.mgr m
```

This is equivalent to

```
SELECT m FROM EmpBean e JOIN e.dept.mgr m
```

Examples: OUTER JOIN

An OUTER JOIN results in a temporary collection that contains combinations of the *left* and *right* operands, subject to the relationship constraints and such that the left operand always appears in R. In the example an outer join results in a temporary collection R that contains department 30, even though the collection **d.emps** is empty. The tuple contains Department 30 along with a NULL value. References to **e** in the query yields a null value.

```
SELECT e FROM DeptBean AS d LEFT OUTER JOIN d.emps AS e
```

The keyword OUTER is optional, as shown here..

```
SELECT e FROM DeptBean AS d LEFT JOIN d.emps AS e
```

You can also use combinations of INNER and OUTER JOIN.

```
SELECT m FROM EmpBean e JOIN e.dept d LEFT JOIN d.mgr m
```

Inheritance in EJB query:

If an Enterprise JavaBeans (EJB) inheritance hierarchy has been defined for an abstract schema, using the abstract schema name in a query statement implies the collection of objects for that abstract schema as well as all subtypes.

Example: Inheritance

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy. The following query returns employees as well as managers and executives:

```
SELECT OBJECT(e) FROM EmpBean e
```

Path expressions:

A path expression is an identification variable followed by the navigation operator (.) and a container managed persistence (CMP) or relationship name.

A path expression that leads to a cmr field can be further navigated if the cmr field is single-valued. If the path expression leads to a multi-valued relationship, then the path expression is terminal and cannot be further navigated. If the path expression leads to a CMP field whose type is a value object, it is possible to navigate to attributes of the value object.

Example: Value object

Assume that `address` is a CMP field for `EmpBean`, which is a value object.

```
SELECT  object(e)  FROM EmpBean e
WHERE  e.address.distance('San Jose') < 10  and e.address.zip = 95037
```

It is best to use the composer pattern to map value object attributes to relational columns if you intend to search on value attributes. If you store value objects in serialized format, then each value object must be retrieved from the database and deserialized. Value object methods can only be done in dynamic queries.

A path expression can also navigate to a bean method. The method must be defined on either the remote or local bean interface. Methods can only be used in dynamic queries. You cannot mix both remote and local methods in a single query statement.

If the query contains remote methods, the dynamic query must be executed using the query remote interface. Using the query remote interface causes the query service to activate beans and create instances of the remote bean interface

Likewise, a query statement with local bean methods must be executed with the query local interface. This causes the query service to activate beans and local interface instances.

Do not use get methods to access CMP and cmr fields of a bean.

If a method has overloaded definitions, the overloaded methods must have different number of parameters.

Methods must have non-void return types and method arguments and return types must be either primitive types byte, short, int, long, float, double, boolean, char or wrapper types from the following list:

Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Date

If any input argument to a method is NULL, it is assumed the method returns a NULL value and the method is not invoked.

A collection valued path expression can be used in the FROM clause as a collection member declaration, and with the IS EMPTY, MEMBER OF, and EXISTS predicates in the WHERE clause.

`FROM EmpBean e WHERE e.dept.mgr.name='Bob'`	OK
`FROM EmpBean e WHERE e.dept.emps.name='BOB'`	INVALID -- cannot navigate through emps because it is multivalued
`FROM EmpBean e,  IN (e.dept.emps) e1` `WHERE e1.name='BOB'`	OK
`FROM EmpBean e WHERE e.dept.emps IS EMPTY`	OK

WHERE clause:

The WHERE clause lists search conditions for items to add to a result set.

The WHERE clause contains search conditions composed of the following:
- literal values
- input parameters
- expressions
- basic predicates
- quantified predicates

- BETWEEN predicate
- IN predicate
- LIKE predicate
- NULL predicate
- EMPTY collection predicate
- MEMBER OF predicate
- EXISTS predicate
- IS OF TYPE predicate

If the search condition evaluates to TRUE, the tuple is added to the result set.

Literals:

Literals can be considered constants that do not change in value.

A string literal is enclosed in single quotes. A single quote that occurs within a string literal is represented by two single quotes. For example: 'Tom''s'. A string literal cannot exceed the maximum length that is supported by the underlying persistent datastore.

A numeric literal can be any of the following:
- an exact value such as 57, -957, +66
- any value supported by Java long
- a decimal literal such as 57.5, -47.02
- an approximate numeric value such as 7E3, -57.4E-2

A decimal or approximate numeric value must be in the range supported by the underlying persistent datastore.

A boolean literal can be the keyword TRUE or FALSE and is case insensitive.

Input parameters:

Input parameters are designated by the question mark followed by a number; for example: `?2`. Input parameters are numbered starting at 1 and correspond to the arguments of the finder or select method; therefore, a query must not contain an input parameter that exceeds the number of input arguments.

An input parameter can be a primitive type of byte, short, int, long, float, double, boolean, char or wrapper types of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Char, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, an EJBObject, or a binary data string in the form of Java byte[].

An input parameter must not have a NULL value. To search for the occurrence of a NULL value the NULL predicate should be used.

Expressions:

An expression specifies a value.

Conditional expressions can consist of comparison operators and logical operators (AND, OR, NOT).

Arithmetic expressions can be used in comparison expressions and can be composed of arithmetic operations and functions, path expressions that evaluate to a numeric value and numeric literals and numeric input parameters.

String expressions can be used in comparison expressions and can be composed of string functions, path expressions that evaluate to a string value and string literals and string input parameters. A CMP field of type char is handled as if it were a string of length 1.

Binary expressions can be used in comparison expressions and can be composed of path expressions that evaluate to the Java byte[] type as well as input parameters of type byte[].

Boolean expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a boolean value and TRUE and FALSE keywords and boolean input parameters.

Reference expressions can be used with = and <> comparison and can be composed of path expressions that evaluate to a cmr field, an identification variable and an input parameter whose type is an EJB reference

Four different expression types are supported for working with date-time types. For portability the java.util.Calendar type should be used. DB2 style date, time and timestamp expressions are supported if the datastore is DB2 and the CMP field is of type java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp. If you use DB2 UDB, you might obtain a syntax error when using the java.sql.Timestamp.ojbect. You must use the syntax TIMESTAMP 'yyyy-mm-dd hh:mm:ss.nnnn'.

A Calendar type can be compared to another Calendar type, an exact numeric literal or input parameter of type long whose value is the standard Java long millisecond value.

The following query finds all employees born before Jan 1, 1990:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e.birthDate <  631180800232
```

Date expressions can be used in comparison expressions and can be composed of operators + - , date duration expressions and date functions, path expressions that evaluate to a date value, string representation of a date and date input parameters.

Time expressions can be used in comparison expressions and can be composed of operators + - , time duration expressions and time functions, path expressions that evaluate to a time value, string representation of time and time input parameters.

Timestamp expressions can be used in comparison expressions and can be composed of operators + - , timestamp duration expressions and timestamp functions, path expressions that evaluate to a timestamp value, string representation of a timestamp and timestamp input parameters.

Standard bracketing ( ) for ordering expression evaluation is supported.

The operators and their precedence order from highest to lowest are:
- Navigation operator (.)
- Arithmetic operators in precedence order:
  - + - unary
  - * / multiply, divide
  - + - add, subtract
- Comparison operators: =, >, <, >=, <=, <>(not equal)
- Logical operator NOT
- Logical operator AND
- Logical operator OR

Null value semantics:

The following describe the semantics of NULL values.
- Comparison or arithmetic operations with an unknown (NULL) value yield an unknown value
- In a Java 2 platform, Enterprise Edition (J2EE) version 1.3 application, a path expression uses an outer-join semantic where a NULL field or cmr value evaluates to NULL. In J2EE version 1.4, the path expression uses an inner-join semantic.
- The IS NULL and IS NOT NULL operators can be applied to path expressions and return TRUE or FALSE. Boolean operators AND, OR and NOT use three valued logic.

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

	NOT
True	False
False	True
Unknown	Unknown

Example: Null value semantics

```
select object(e) from EmpBean where e.salary > 10  and e.dept.budget > 100
```

If salary is NULL the evaluation of `e.salary > 10` returns unknown and the employee object is not returned. If the cmr field dept or budget is NULL evaluation of `e.dept.budget > 100` returns unknown and the employee object is not returned.

```
select object(e) from EmpBean where e.dept.budget is null
```

In J2EE 1.3 if dept or budget is NULL evaluation of `e.dept.budget` is null returns TRUE and the employee object is returned. In J2EE 1.4 the employee object is returned only if budget is NULL.

```
select object(e) from EmpBean e ,  in (e.dept.emps) e1 where e1.salary > 10
```

If dept is NULL, then the multivalued path expression `e.dept.emps` results in an empty collection (not a collection that contains a NULL value). An employee with a null dept value will not be returned.

```
select object(e) from EmpBean e where  e.dept.emps is empty
```

If dept is NULL the evaluation of the predicate in unknown and the employee object is not returned.

```
select object(e) from EmpBean e , EmpBean e1  where e member of  e1.dept.emps
```

If dept is NULL evaluation of the member of predicate returns unknown and the employee is not returned.

Date time arithmetic and comparisons:

DATE, TIME and TIMESTAMP values can be compared with another value of the same type. Comparisons are chronological. Date time values can also be incremented, decremented, and subtracted.

If the datastore is DB2, then DB2 string representation of DATE, TIME and TIMESTAMP types can also be used. A string representation of a date or time can use ISO, USA, EUR or JIS format. A string representation of a timestamp uses ISO format.

Format	Date format	Date examples	Time format	Time examples
ISO	yyyy-mm-dd	1987-02-24 1987-2-24	hh.mm.ss	13.50.00 13.50
USA	mm/dd/yyyy	2/24/1987	hh:mm AM or PM	1:50 pm 02:10 AM
EUR	dd.mm.yyyy	24.02.1987 24.2.1987	hh.mm.ss	13.50.00 13.55

Format	Date format	Date examples	Time format	Time examples
JIS	yyyy-mm-dd	1987-02-24	hh:mm:ss	13:50 13:50:05

Example 1: Date time arithmetic comparisons

```
e.hiredate > '1990-02-24'
```

The timestamp of February 24th, 1990 1:50 pm can be represented as follows:

```
'1990-02-24-13.50.00.000000'  or
'1990-02-24-13.50.00'
```

If the datastore is DB2, DB2 decimal durations can be used in expressions and comparisons. A date duration is a decimal(8,0) number that represents the difference between two dates in the format YYYYMMDD. A time duration is a decimal(6,0) number that represents the difference between two time values as HHMMSS. A timestamp duration is a decimal(20,6) number representing the differences between two timestamp values as YYYYMMDDHHMMSS.ZZZZZZ (ZZZZZZ is the number of microseconds and is to the right of the decimal point) .

Two date values (or time values or timestamp values) can be subtracted to yield a duration. If the second operand is greater than the first the duration is a negative decimal number. A duration can be added or subtracted from a datetime value to yield a new datetime value.

Example 2: Date time arithmetic comparisons

`DATE('3/15/2000') - '12/31/1999'` results in a decimal number 215 which is a duration of 0 years, 2 months and 15 days.

Durations are really decimal numbers and can be used in arithmetic expressions and comparisons.

`( DATE('3/15/2000') - '12/31/1999' ) + 14 > 215` evaluates to TRUE.

`DATE('12/31/1999') + DECIMAL(215,8,0)` results in a date value 3/15/2000.

`TIME('11:02:26') - '00:32:56'` results in a decimal number 102930 which is a time duration of 10 hours, 29 minutes and 30 seconds.

`TIME('00:32:56') + DECIMAL(102930,6,0)` results in a time value of 11:02:26.

`TIME('00:00:59') + DECIMAL(240000,6,0)` results in a time value of 00:00:59.

`e.hiredate + DECIMAL(500,8,0) > '2000-10-01'` means compare the hiredate plus 5 months to the date 10/01/2000.

Basic predicates:

A basic predicate compares two values.

Basic predicates can be of two forms, for example:

```
expression-1  comparison-operator  expression-2
expression-3  comparison-operator ( subselect )
```

The subselect must not return more than one value and the subselect cannot return a type of an Enterprise JavaBean (EJB) reference. Boolean types and reference types only support = and <> comparisons.

Example: Basic predicates

```
d.name='Java Development'
e.salary > 20000
e.salary > ( select avg(e.salary) from EmpBean e)
```

Quantified predicates:

A quantified predicate compares a value with a set of values produced by a subselect.

Use the syntax:

```
expression   comparison-operator   SOME  |  ANY  |   ALL    ( subselect )
```

The expression must not evaluate to a reference type.

When SOME or ANY is specified the result of the predicate is as follows:
- TRUE if the comparison is true for at least one value returned by the subselect.
- FALSE if the subselect is empty or if the comparison is false for every value returned by the subselect.
- UNKNOWN if the comparison is not true for all of the values returned by the subselect and at least one of the comparisons is unknown because of a null value.

When ALL is specified the result of the predicate is as follows:
- TRUE if the subselect returns empty or if the comparison is true to every value returned by the subselect.
- FALSE if the comparison is false for at least one value returned by the subselect.
- UNKNOWN if the comparison is not false for all values returned by the subselect and at least one comparison is unknown because of a null value.

BETWEEN predicate:

The BETWEEN predicate determines whether a given value lies between two other given values.

The syntax for the predicate is:

```
expression  [NOT]  BETWEEN expression-2  AND expression-3
```

The expression must not evaluate to a boolean or reference type.

Example: BETWEEN predicate

```
e.salary BETWEEN 50000 AND 60000
```

is equivalent to:

```
e.salary >= 50000  AND e.salary <= 60000
e.name NOT BETWEEN 'A' AND 'B'
```

is equivalent to:

```
e.name < 'A'  OR e.name > 'B'
```

IN predicate:

The IN predicate compares a value to a set of values.

It can have one of two forms:

```
expression [NOT] IN  ( subselect )
expression [NOT] IN  ( value1, value2,  .... )
```

ValueN can either be a literal value or an input parameter. The expression cannot evaluate to a reference type.

Example: IN predicate

```
e.salary IN ( 10000, 15000 )
```

is equivalent to

```
( e.salary = 10000  OR  e.salary = 15000 )
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

LIKE predicate:

The LIKE predicate searches a string value for a certain pattern.

The syntax for this predicate is:

```
string-expression   [NOT]  LIKE  pattern    [ ESCAPE escape-character ]
```

The pattern value is a string literal or parameter marker of type string in which the underscore (_) stands for any single character and percent (%) stands for any sequence of characters (including empty sequence). Any other character stands for itself. The escape character can be used to search for character _ and %. The escape character can be specified as a string literal or an input parameter.

If the string-expression is null, then the result is unknown.

If both string-expression and pattern are empty, then the result is true.

Example: LIKE predicate

- '' LIKE '' is true
- '' LIKE '%' is true
- e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
- e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
- e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
- e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
- e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'

NULL predicate:

The NULL predicate tests for null values.

Use the syntax:

```
single-valued-path-expression IS [NOT] NULL
```

Example: NULL predicate

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

EMPTY collection predicate:

You can use the EMPTY collection predicate to test if a multivalued relationship has no members.

Use the following syntax:

```
collection-valued-path-expression  IS [NOT]  EMPTY
```

Example: Empty collection predicate

To find all departments with no employees:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

MEMBER OF predicate:

This expression tests whether the object reference specified by the single valued path expression or input parameter is a member of the designated collection.

If the collection valued path expression designates an empty collection the value of the MEMBER OF expression is FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ] collection-valued-path-expression
```

Example: MEMBER OF predicate

Find employees that are not members of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Find employees whose manager is a member of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

EXISTS predicate:

The exists predicate tests for the presence or absence of a condition specified by a subselect.

Use the syntax:

```
EXISTS ( subselect )
EXISTS collection-valued-path-expression
```

The result of EXISTS is true if the subselect returns at least one value or the path expression evaluates to a nonempty collection, otherwise the result is false.

To negate an EXISTS predicate, precede it with the logical operator NOT.

Example: EXISTS predicate

Return departments that have at least one employee earning more than 1000000:

```
SELECT  OBJECT(d) FROM  DeptBean d
WHERE EXISTS ( SELECT  1  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Return departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT 1 FROM IN (d.emps) e)
```

The above query can also be written as follows:

```
SELECT OBJECT(d) FROM DeptBean d WHERE NOT EXISTS d.emps
```

IS OF TYPE predicate:

The IS OF TYPE predicate is used to test the type of an Enterprise JavaBeans (EJB) reference. It is similar in function to the Java instance of operator.

IS OF TYPE is used when several abstract beans have been grouped into an EJB inheritance hierarchy. The type names specified in the predicate are the bean abstract names. The ONLY option can be used to specify that the reference must be exactly this type and not a subtype.

```
identification-variable IS OF TYPE ( [ONLY] type-1, [ONLY] type-2, ..... )
```

Example: IS OF TYPE predicate

Suppose that bean `ManagerBean` is defined as a subtype of `EmpBean` and `ExecutiveBean` is a subtype of `ManagerBean` in an EJB inheritance hierarchy.

The following query returns employees as well as managers and executives:

```
SELECT  OBJECT(e) FROM EmpBean e
```

If you are interested in objects which are employees and not managers and not executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ONLY EmpBean )
```

If you are interested in object which are managers or executives:

```
SELECT OBJECT(e) FROM EmpBean e WHERE e IS OF TYPE( ManagerBean)
```

The above query is equivalent to the following query:

```
SELECT  OBJECT(e) FROM ManagerBean e
```

If you are interested in managers only and not executives:

```
SELECT  OBJECT(e) FROM  EmpBean e WHERE e IS OF TYPE( ONLY ManagerBean)
```

or:

```
SELECT  OBJECT(e) FROM  ManagerBean e
WHERE  e IS OF TYPE (ONLY ManagerBean)
```

Scalar functions:

An Enterprise JavaBeans (EJB) query contains scalar functions for doing type conversions, string manipulation, and for manipulating date-time values.

The list of scalar functions is documented in the topic EJB query: Scalar functions.

Example: Scalar functions

Find employees hired in 1999:

```
SELECT OBJECT(e) FROM EmpBean e where YEAR(e.hireDate) = 1999
```

The only scalar functions that are guaranteed to be portable across backend datastore vendors are the following:

- ABS
- MOD
- SQRT
- CONCAT
- LENGTH

- LOCATE
- SUBSTRING
- UCASE
- LCASE

The other scalar functions should be used only when DB2 is the backend datastore.

EJB query: Scalar functions:

Enterprise JavaBeans (EJB) query contains scalar built-in functions for doing type conversions, string manipulation, and for manipulating date-time values.

EJB query scalar built-in functions are listed below:

Numeric functions

```
ABS ( < any numeric datatype > ) -> < any numeric datatype >
MOD ( <int>, <int> ) -> int
SQRT ( < any numeric datatype > ) -> Double
```

Type conversion functions

```
CHAR ( < any  numeric datatype > ) ->  string
CHAR ( <  string  > ) ->  string
CHAR ( < any datetime  datatype >  [, Keyword k ]) ->  string
```

Datetime datatype is converted to its string representation in a format specified by the keyword k. The valid keywords values are ISO, USA, EUR or JIS. If k is not specified the default is ISO.

```
BIGINT ( < any numeric datatype > ) -> Long
BIGINT ( < string > ) -> Long
```

The function in the second line of the following code converts the argument to an integer n by truncation, and returns the date that is n-1 days after January 1, 0001:

```
DATE ( < date string > ) -> Date
DATE (  < any numeric datatype>) -> Date
```

The following function returns date portion of a timestamp:

```
DATE( timestamp ) -> Date
DATE ( < timestamp-string > ) -> Date
```

The following function converts number to decimal with optional precision p and scale s.

```
DECIMAL ( < any numeric datatype > [, p [ ,s ] ] ) -> Decimal
```

The following function converts string to decimal with optional precision p and scale s.

```
DECIMAL ( < string > [ , p [ , s ] ] ) -> Decimal
DOUBLE ( < any numeric datatype > ) -> Double
DOUBLE ( < string > ) -> Double
FLOAT ( < any numeric datatype > ) -> Double
FLOAT ( < string > ) -> Double
```

Float is a synonym for DOUBLE.

```
INTEGER ( < any numeric datatype > ) -> Integer
INTEGER ( < string > ) -> Integer
REAL ( < any numeric datatype > ) -> Float
SMALLINT ( < any numeric datatype ) -> Short
SMALLINT ( < string > ) -> Short
```

```
TIME ( < time > ) -> Time
TIME ( < time-string  > ) -> Time
TIME ( < timestamp > ) -> Time
TIME ( < timestamp-string  > ) -> Time

TIMESTAMP ( < timestamp > ) -> Timestamp
TIMESTAMP ( < timestamp-string > ) -> Timestamp
```

String functions

```
CONCAT ( <string>, <string>  ) -> String
```

The following function returns a character string representing absolute value of the argument not including its sign or decimal point. For example, `digits( -42.35)` is ″4235″.

```
DIGITS ( Decimal d  ) -> String
```

The following function returns the length of the argument in bytes. If the argument is a numeric or datetime type, it returns the length of internal representation.

```
LENGTH ( < string >  ) -> Integer
```

The following function returns a copy of the argument string where all upper case characters have been converted to lower case.

```
LCASE ( < string > ) -> String
```

The following function returns the starting position of the first occurrence of argument 1 inside argument 2 with optional start position. If not found, it returns 0.

```
LOCATE ( String s1 , String s2  [, Integer start ] ) -> Integer
```

The following function returns a substring of s beginning at character m and containing n characters. If n is omitted, the substring contains the remainder of string s. The result string is padded with blanks if needed to make a string of length n.

```
SUBSTRING ( String s ,  Integer m [ , Integer n ] ) -> String
```

The following function returns a copy of the argument string where all lower case characters have been converted to upper case.

```
UCASE ( < string > ) -> String
```

Date - time functions

The following function returns the day portion of its argument. For a duration, the return value can be -99 to 99.

```
DAY (  Date ) ->  Integer
DAY ( < date-string > ) ->  Integer
DAY ( < date-duration > ) -> Integer
DAY ( Timestamp  ) ->  Integer
DAY ( < timestamp-string > ) ->  Integer
DAY ( < timestamp-duration > ) -> Integer
```

The following function returns one more than number of days from January 1, 0001 to its argument.

```
DAYS ( Date  ) ->  Integer
DAYS ( < Date-string > ) ->  Integer
DAYS ( Timestamp  ) ->  Integer
DAYS ( < timestamp-string > ) ->  Integer
```

The following function returns the hour part of its argument. For a duration, the return value can be -99 to 99.

```
HOUR ( Time ) -> Integer
HOUR ( < time-string > ) -> Integer
HOUR ( < time-duration > ) ->  Integer
HOUR ( Timestamp ) -> Integer
HOUR ( < timestamp-string > ) -> Integer
HOUR ( < timestamp-duration >  ) -> Integer
```

The following function returns the microsecond part of its argument.

```
MICROSECOND ( Timestamp ) -> Integer
MICROSECOND ( < timestamp-string > ) -> Integer
MICROSECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the minute part of its argument. For a duration, the return value can be -99 to 99.

```
MINUTE ( Time ) -> Integer
MINUTE ( < time-string > ) -> Integer
MINUTE ( < time-duration > ) ->  Integer
MINUTE ( Timestamp ) -> Integer
MINUTE ( < timestamp-string > ) -> Integer
MINUTE ( < timestamp-duration >  ) -> Integer
```

The following function returns the month portion of its argument. For a duration, the return value can be -99 to 99.

```
MONTH (  Date ) ->  Integer
MONTH ( < date-string > ) ->  Integer
MONTH ( < date-duration > ) -> Integer
MONTH ( Timestamp  ) ->  Integer
MONTH ( < timestamp-string > ) ->  Integer
MONTH ( < timestamp-duration > ) -> Integer
```

The following function returns the second part of its argument. For a duration, the return value can be -99 to 99.

```
SECOND ( Time ) -> Integer
SECOND ( < time-string > ) -> Integer
SECOND ( < time-duration > ) ->  Integer
SECOND ( Timestamp ) -> Integer
SECOND ( < timestamp-string > ) -> Integer
SECOND ( < timestamp-duration >  ) -> Integer
```

The following function returns the year portion of its argument. For a duration, the return value can be -9999 to 9999.

```
YEAR (  Date ) ->  Integer
YEAR ( < date-string > ) ->  Integer
YEAR ( < date-duration > ) -> Integer
YEAR ( Timestamp  ) ->  Integer
YEAR ( < timestamp-string > ) ->  Integer
YEAR ( < timestamp-duration > ) -> Integer
```

Aggregation functions:

Aggregation functions operate on a set of values to return a single scalar value. You can use these functions in the select and subselect methods.

The following example illustrates an aggregation:

```
SELECT  SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

This aggregation computes the total salary for department 20.

The aggregation functions are AVG, COUNT, MAX, MIN, and SUM. The syntax of an aggregation function is illustrated in the following example:

```
aggregation-function  (    [ ALL |  DISTINCT ]  expression )
```

or:

```
COUNT( [ ALL |  DISTINCT ] identification-variable )
```

or:

```
COUNT( * )
```

The DISTINCT option eliminates duplicate values before applying the function. ALL is the default option and does not eliminate duplicates. Null values are ignored in computing the aggregate function except in the cases of COUNT(*) and COUNT(identification-variable), which return a count of all the elements in the set.

If your datastore is Informix, you must limit the expression argument to a single valued path expression when using the COUNT function or the DISTINCT forms of the functions SUM, AVG, MIN, and MAX.

Defining return type

For a select method using an aggregation function, you can define the return type as a primitive type or a wrapper type. The return type must be compatible with the return type from the datastore. The MAX and MIN functions can apply to any numeric, string or datetime datatype and return the corresponding datatype. The SUM and AVG functions take a numeric type as input, and return the same numeric type that is used in the datastore. The COUNT function can take any datatype, and returns an integer.

When applied to an empty set, the SUM, AVG, MAX, and MIN functions can return a null value. The COUNT function returns zero (0) when it is applied to an empty set. Use wrapper types if the return value might be NULL; otherwise, the container displays an ObjectNotFound exception.

Using GROUP BY and HAVING

The set of values that is used for the aggregate function is determined by the collection that results from the FROM and WHERE clause of the query. You can divide the set into groups and apply the aggregation function to each group. To perform this action, use a GROUP BY clause in the query. The GROUP BY clause defines grouping members, which comprise a list of path expressions. Each path expression specifies a field that is a primitive type of byte, short, int, long, float, double, boolean, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time or java.sql.Timestamp.

The following example illustrates the use of the GROUP BY clause in a query that computes the average salary for each department:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

In division of a set into groups, a NULL value is considered equal to another NULL value.

Just as the WHERE clause filters tuples (that is, records of the return collection values) from the FROM clause, the groups can be filtered using a HAVING clause that tests group properties involving aggregate functions or grouping members:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING  COUNT(*) > 3  AND  e.dept.deptno > 5
```

This query returns the average salary for departments that have more than three employees and the department number is greater than five.

It is possible to use a HAVING clause without a GROUP BY clause, in which case the entire set is treated as a single group, to which the HAVING clause is applied.

SELECT clause:

The SELECT clause consists of either a single identification variable that is defined in the FROM clause, or a single valued path expression that evaluates to an object reference or container managed persistence (CMP) value. You can use the DISTINCT keyword to eliminate duplicate references.

For finder and select queries, the syntax of the SELECT clause is illustrated in the following example:

```
SELECT  [  ALL | DISTINCT  ]
 { single-valued-path-expression  |  aggregation expression  |  OBJECT ( identification-variable )  }
```

For a query that defines a finder method, the query must return an object type consistent with the home that is associated with the finder method. For example, a finder method for a department home can not return employee objects.

Example: SELECT clause

Find all employees that earn more than John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean e
WHERE  ej.name = 'John'  and e.salary > ej.salary
```

Find all departments that have one or more employees who earn less than 20000:

```
SELECT DISTINCT  e.dept  FROM EmpBean e where e.salary < 20000
```

A select method query can have a path expression that evaluates to an arbitrary value:

```
SELECT  e.dept.name  FROM EmpBean e where e.salary < 2000
```

The previous query returns a collection of name values for those departments having employees earning less than 20000.

A select method query can return an aggregate value:

```
SELECT  avg(e.salary)  FROM EmpBean e
```

ORDER BY clause:

The ORDER BY clause specifies an ordering of the objects in the result collection

Use the syntax:

```
ORDER BY  [ order_element ,]* order_element
order_element ::= { path-expression | integer } [ ASC | DESC ]
```

The path expression must specify a single valued field that is a primitive type of byte, short, int, long, float, double, char or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Calendar, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp.

ASC specifies ascending order and is the default. DESC specifies descending order.

Integer refers to a selection expression in the SELECT clause.

Example: ORDER BY clause

Return department objects in decreasing deptno order:

```
SELECT  OBJECT(d) FROM  DeptBean d ORDER BY  d.deptno DESC
```

Return employee objects sorted by department number and name:

```
SELECT  OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC,  e.name DESC
```

UNION operation:

The UNION clause specifies a combination of the output of two subqueries. The two queries must return the same number of elements and compatible types.

For the purposes of UNION, all Enterprise JavaBeans (EJB) types in the same inheritance hierarchy are considered compatible. UNION requires that equality be defined for the element types.

```
query_expression := query_term [UNION [ALL] query_term]*

query_term :=  {select_clause_dynamic  from_clause  [where_clause]
 [group_by_clause] [having_clause] } | (query_expression) }
```

You cannot use dependent value objects with UNION.

UNION ALL combines all results together in a single collection.

UNION combines results but eliminates duplicates.

If ORDER BY is used together with UNION, the ORDER BY must refer to selection expression using integer numbers.

Examples: UNION operation

This example returns a collection of all employee objects of type EmpBean and all manager objects of type ManagerBean where ManagerBean is a subtype of EmpBean.

```
select e from EmpBean e union all select m from DeptBean d, in(d.mgr) m
```

This example shows a query that is not valid, because EmpBean and DeptBean are not compatible.

```
select e from EmpBean e union all select d from DeptBean d
```

Subqueries:

A subquery can be used in quantified predicates, the EXISTS predicate, or the IN predicate. A subquery should only specify a single element in the SELECT clause.

When a path expression appears in a subquery, the identification variable of the path expression must be defined either in the subquery, in one of the containing subqueries, or in the outer query. A scalar subquery is a subquery that returns one value. A scalar subquery can be used in a basic predicate and in the SELECT clause of a dynamic query.

Example: Subqueries

```
SELECT  OBJECT(e) FROM  EmpBean e
WHERE e.salary > ( SELECT  AVG(e1.salary) FROM  EmpBean e1)
```

The above query returns employees who earn more than average salary of all employees.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE  e.salary >
( SELECT AVG(e1.salary) FROM  IN  (e.dept.emps) e1  )
```

The above query returns employees who earn more than average salary of their department.

```
SELECT  OBJECT(e) FROM EmpBean e WHERE e.salary =
( SELECT MAX(e1.salary) FROM IN (e.dept.emps) e1  )
```

The above query returns employees who earn the most in their department.

```
 SELECT OBJECT(e) FROM EmpBean e
WHERE e.salary > ( SELECT AVG(e.salary) FROM EmpBean e1
WHERE YEAR(e1.hireDate) =  YEAR(e.hireDate)  )
```

The above query returns employees who earn more than the average of employees hired in same year.

EJB query language limitations and restrictions:

When using the Enterprise JavaBeans (EJB) query language on the product, deviations can be seen in comparison to standard EJB query language. The limitations and restrictions you must be aware of are listed in the following section.

This topic outlines current known limitations and restrictions.

- EJB query language (QL) queries involving enterprise beans with keys made up of relationships to other enterprise beans appear as not valid and cause errors at deployment time. This is a known problem.
- The IBM EJB QL support extends the EJB 2.0 specification in various ways, including relaxing some restrictions, adding support for more DB2 functions, and so on. If portability across various vendor databases or EJB deployment tools is a concern, then care should be taken to write all EJB QL queries strictly according to Chapter 11 in the EJB 2.0 specification.
- Pre-loading across m:n relationships results in the generation of inaccurate structured query language (SQL). This is a known limitation that may be addressed in the future.
- Pre-loading across self referencing relationships causes inaccurate SQL to be generated.
- Avoid relationships between parent and children enterprise beans within the same inheritance hierarchy that are not well-defined.
- EJB Query Language validation for EJB 2.0 JAR files currently runs as a part of the EJB-RDB Mapping validation. If a mapping document (Map.mapxmi file) does not exist in the project, the EJB queries are not validated.

EJB query compatibility issues with SQL:

Because an Enterprise JavaBeans (EJB) query is compiled into structured query language (SQL), you must be aware of compatibility issues between the Java language and SQL.

The two languages differ along the following points that can be critical to correct EJB query formulation:

- The comparison semantics of SQL strings do not exactly match those of the Java language. For example: `'A'` (the letter A) and`'A '` (the letter A plus a blank space) are considered equal in SQL, but not in the Java language.
- Comparisons and collating order depend on the underlying database. For example, if you are using DB2 with an EBCDIC code page, the collating order is not the same as doing the sort in a Java program. Some databases sort the NULL value low while others sort the NULL value high.
- An arithmetic overflow causes an exception in SQL, but not in the Java language.
- SQL databases have differing minimum and maximum ranges for floating point values, which can differ from floating point value ranges in the Java language. Values near the range limits of Java Double may fail to translate into SQL.
- Java methods do not translate into SQL; therefore standard EJB queries cannot include Java methods.

  Note: Only with the dynamic EJB query service can you use functions that do not translate into SQL. Such functions include Java methods and converters or composers that are used in mapping enterprise beans to relational databases (RDBs). A standard finder or select query that uses any of these functions fails at deployment time with the message ″Cannot push down query″. (You can resolve this problem by changing either the query or the mapping.) The dynamic query run time, however, processes the query by performing the operation involving the function in the application server.

Database restrictions for EJB query:

The Enterprise JavaBeans (EJB) query functions must adhere to certain restrictions for databases.

General database restriction

- All of the enterprise beans involved in a given query must map to the same data source. The EJB query does not support cross-data source join operations.
- It is possible that a structured query language (SQL) statement generated by the WebSphere Application Server deployment code generation utility for an *ejbSelect* Enterprise JavaBeans query language query returns rows in a result set that consist of null values in all columns.

  During run time persistence manager saves the set received as a result from this query. When your application retrieves the primary key of the result bean, persistence manager calls the extractor. The extractor is a method that is an EJB deploy generated class. This method returns a value of **0** for any null column entries. This value is passed back to the EJB container to forward to the application. The EJB container invokes the bean instance with the PK value of **0**. This could create a problem, as the end user cannot determine if this bean instance has a *null PK* or a *PK value of 0*.

  To avoid this, use the *IS NOT NULL* clause in the finder query to eliminate such null values from the result set.

Specific database restrictions

Different database products place different restrictions on elements that can be included in EJB query statements. Following is a list of those restrictions; check with your database administrator to see if any apply in your environment:

- Certain functions are used in queries that run against DB2 only, because these functions are not supported by other databases. These functions include date and time arithmetic expressions, certain scalar functions (those *not* listed as portable across vendors), and implied scalar functions when used for mapping certain CMP fields. For example, consider mapping an int numeric type to a decimal (5,2) type field. When deployed against a database other than DB2, a finder or select query that contains a CMP field with this particular mapping fails, producing a `Cannot push down query` error message.
- A CMP of type String, when mapped to a character large object (CLOB) in the database, cannot be used in comparison operations because the database does not support CLOB comparisons.
- Databases can impose limits on the length of string values that are used either as literals or input parameters with comparison operators. These limits can hinder query performance. For example: For DB2 on the z/OS platform, the search ″name = ?1″ can fail if the value of ?1 at run time is greater than 255 in length.
- Mapping a numeric CMP type to a column that contains a dissimilar type can cause unexpected results. For example, consider the case of mapping the int numeric type to a column of type decimal (5,2). This scenario does not preserve an exact decimal value (for example, the value 12.25) over the course of transfer from the database to the enterprise bean CMP field, and back again to the database. This mapping causes replacement of the initial value with a whole number (in this case, 12). Consequently, you want to avoid using the CMP field in comparison operations when the CMP field uses a mapping of this nature.
- Some databases do not support a datatype that corresponds to the semantics of java.sql.Time. For example: If a CMP field of type java.sql.Time is mapped to an Oracle DATE column, comparisons on time might not produce the expected result because the year-month-day portion of the column value is truncated in the mapping.
- Some databases treat a zero length string value ('') as a null value; this approach can affect the query results. For the sake of portability, avoid the use of zero length string values.
- Some databases perform division between two integer values using integer arithmetic rules, while others use non-integer rules. This discrepancy might not be desirable in environments that use both kinds of databases. For the sake of portability, avoid the division of integer values in an EJB query.
- Current releases of UDB DB2 for i5/OS only support a TIMESTAMP value of the format 'yyyy-mm-dd-hh.mm.ss.nnnnnn'. This is not compatible with the standard format supported by the java.sql.Timestamp class, which is 'yyyy-mm-dd-hh mm.ss.nnnnnn'. The TIMESTAMP scalar function should be used to convert a string representation of a java.sql.Timestamp object to a value that can be recognized by DB2 UDB for i5/OS.

Rules for data type manipulation in EJB query:

When using an Enterprise JavaBean (EJB) query to work with data types, certain rules must be followed.

Rules for container managed persistence (CMP) field type

You can use a CMP field of any type in a SELECT clause. You must, however, use fields of only the following types in search conditions and in grouping or ordering operations:

- Primitive types: byte, short, int, long, float, double, boolean, char
- Object types: Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Calendar, java.util.Date
- JDBC types: java.sql.Date, java.sql.Time, java.sql.Timestamp
- Binary string: byte[]

Converters and basic types

If ALL of the following conditions occur:

- a CMP field of one of the basic types listed previously is mapped to an SQL column using a converter
- the CMP field appears in the left hand side of a basic predicate
- the right hand side of the predicate is a literal or input parameter

then the toData() method of the converter is used to compute the SQL search value.

For example, given a converter that maps the integer value 10 to the string value ″Ten,″ the following EJB query:

```
e.cmp = 10
```

is translated into the following SQL query:

```
column = 'Ten'
```

If you include a more complicated predicate, such as in the following example:

```
e.cmp * 10 >  e.salary
```

in a finder or select query, you receive the `Cannot push down query` error message. Use the dynamic EJB query service for such multi-function queries; the dynamic query run time processes the predicate in the application server.

Overall, converters preserve equality, collating sequence, and NULL values. If a converter does not meet these requirements, avoid using it for CMP field comparison operations.

User types, converters, and composers

A user type cannot be used in a comparison operation or expression. You can, however, use subfields of the user type in a path expression. For example, consider the CMP addr field with the type `com.exam.Address`, and street, city, and state subfields. The following syntax for a query on this CMP field is not valid:

```
e.addr =  ?1
```

However, a query that designates one of the subfields is valid:

```
e.addr.street = ?1
```

A CMP field can be mapped to an SQL column using Java serialization. Using the CMP field in predicates or expressions for deployment queries usually results in the `Cannot push down query` error message. The dynamic query run time processes the expression by reading and deserializing all instances of the user type in the application server.

However, this expensive process sacrifices performance. You can maintain performance by using a composer in a deployment EJB query. In the previous example, if you want to map the addr field to a binary type, you use a composer to map each subfield to a binary column in the database.

EJB query: Reserved words:

The following words are reserved in WebSphere Application Server Enterprise JavaBeans (EJB) queries.

all, as, distinct, empty, false, from, group, having, in, is, like, select, true, union, where

Avoid using identifiers that start with underscore (for example, _integer) as these are also reserved.

EJB query: BNF syntax:

The Backus-Naur Form (BNF) is one of the most commonly used notations for specifying the syntax of programming languages or command sets. This article lists the syntax for Enterprise JavaBeans (EJB) query language.

```
EJB QL ::= [select_clause] from_clause [where_clause] [order_by_clause]

DYNAMIC EJB QL := query_expression  [order_by_clause]

query_expression := query_term [UNION [ALL] query_term]*

query_term :=  {select_clause_dynamic  from_clause  [where_clause]
 [group_by_clause] [having_clause] } | (query_expression) }  [order_by_clause]


from_clause::=FROM identification_variable_declaration
               [, {identification_variable_declaration | collection_member_declaration } ]*

identification_variable_declaration::=collection_member_declaration |
       range_variable_declaration [join]*

join :=  [ { LEFT [OUTER] | INNER  }]  JOIN  {collection_valued_path_expression | single_valued_path_expression}
 [AS] identifier

collection_member_declaration::=
       IN ( collection_valued_path_expression ) [AS] identifier

range_variable_declaration::=abstract_schema_name [AS] identifier

single_valued_path_expression ::=
       {single_valued_navigation | identification_variable}. ( cmp_field |
 method  |  cmp_field.value_object_attribute |  cmp_field.value_object_method )
          | single_valued_navigation

single_valued_navigation::=
       identification_variable.[  single_valued_cmr_field.  ]*
            single_valued_cmr_field

collection_valued_path_expression ::=
       identification_variable.[  single_valued_cmr_field.  ]*
           collection_valued_cmr_field

select_clause::= SELECT { ALL | DISTINCT }  {single_valued_path_expression |
                        identification_variable |  OBJECT ( identification_variable) |
        aggregate_functions }

select_clause_dynamic ::= SELECT { ALL | DISTINCT }  [ selection , ]*  selection

selection  ::= { expression  |  subselect }  [[AS] id ]

order_by_clause::= ORDER BY [ {single_valued_path_expression | integer}  [ASC|DESC],]*
      {single_valued_path_expression | integer}[ASC|DESC]
```

```
where_clause::= WHERE conditional_expression

conditional_expression ::= conditional_term |
                           conditional_expression OR conditional_term
conditional_term ::= conditional_factor |
                     conditional_term AND conditional_factor
conditional_factor ::= [NOT] conditional_primary
conditional_primary::=simple_cond_expression | (conditional_expression)

simple_cond_expression ::= comparison_expression | between_expression |
       like_expression | in_expression | null_comparison_expression |
       empty_collection_comparison_expression | quantified_expression |
       exists_expression | is_of_type_expression  |  collection_member_expression

between_expression ::= expression [NOT] BETWEEN expression AND expression

in_expression ::= single_valued_path_expression [NOT] IN
           { (subselect) |  ( [ atom ,]* atom )  }

atom = { string-literal | numeric-constant | input-parameter }

like_expression ::= expression [NOT] LIKE
                {string_literal | input_parameter}
                [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::=
       single_valued_path_expression IS [ NOT ] NULL

empty_collection_comparison_expression ::=
       collection_valued_path_expression IS [NOT] EMPTY

collection_member_expression  ::=
       { single_valued_path_expression |  input_paramter }  [ NOT ] MEMBER [ OF ]
         collection_valued_path_expression

quantified_expression ::=
       expression comparison_operator  {SOME | ANY | ALL} (subselect)

exists_expression ::= EXISTS {collection_valued_path_expression |  (subselect)}

subselect ::= SELECT [{ ALL | DISTINCT }]  expression  from_clause [where_clause]
       [group_by_clause] [having_clause]

group_by_clause::= GROUP BY [single_valued_path_expression,]*
                    single_valued_path_expression

having_clause ::= HAVING conditional_expression


is_of_type_expression ::= identifier  IS OF TYPE
            ([[ONLY] abstract_schema_name,]* [ONLY] abstract_schema_name)

comparison_expression ::=  expression   comparison_operator { expression |  ( subquery ) }

comparison_operator ::=    = | > | >= | < | <= | <>

method ::=  method_name( [[expression , ]* expression ] )

expression ::= term |   expression {+|-} term

term ::=  factor |  term {*|/} factor

factor ::= {+|-} primary

primary ::= single_valued_path_expression | literal |
       ( expression ) |  input_parameter | functions | aggregate_functions
```

```
aggregate_functions :=
       AVG([ALL|DISTINCT] expression) |
       COUNT({[ALL|DISTINCT] expression | * | identification_variable }) |
       MAX([ALL|DISTINCT] expression) |
       MIN([ALL|DISTINCT] expression) |
       SUM([ALL|DISTINCT] expression) |

functions ::=
        ABS(expression) |
        BIGINT(expression) |
        CHAR({expression [,{ISO|USA|EUR|JIS}] )  |
        CONCAT (expression , expression ) |
        DATE(expression) |
        DAY({expression ) |
        DAYS( expression) |
        DECIMAL( expression [,integer[,integer]])
        DIGITS( expression) |
        DOUBLE( expression ) |
        FLOAT( expression) |
        HOUR ( expression ) |
        INTEGER( expression ) |
        LCASE ( expression) |
        LENGTH(expression) |
        LOCATE( expression, expression [, expression] ) |
        MICROSECOND( expression ) |
        MINUTE ( expression ) |
        MOD ( expression , expression ) |
        MONTH( expression ) |
        REAL( expression) |
        SECOND( expression ) |
        SMALLINT( expression )  |
        SQRT (  expression) |
        SUBSTRING( expression, expression[, expression]) |
        TIME( expression ) |
        TIMESTAMP( expression ) |
        UCASE ( expression) |
        YEAR( expression )

xrel := XREL identification_variable . { single_valued_cmr_field | collection_valued_cmr_field }
             [ , identification_variable . { single_valued_cmr_field | collection_valued_cmr_field } ]*
```

EJB specification and WebSphere query language comparison:

WebSphere Application Server extends the Enterprise JavaBeans (EJB) query language with elements of its own.

WebSphere Application Server supports the following extensions to the EJB query language.

Item	
Delimited identifiers	
Dependent Value object attributes used in path expressions	
EJB Inheritance	
EXISTS predicate	
Java methods: EJB bean methods or value object methods	dynamic query only
Multiple element select clauses	dynamic query only
SQL Date/time expressions	
Subqueries, group by, and having clauses	

Using the dynamic query service

There are times in the development process when you might prefer to use the dynamic query service rather than the regular Enterprise JavaBean (EJB) query service (which can be referred to as *deployment query*). During testing, for instance, the dynamic query service can be used at application run time, so you do not have to re-deploy your application.

About this task

Following are common reasons for using the dynamic query service rather than the regular EJB query service:

- You need to programmatically define a query at application run time, rather than at deployment.
- You need to return multiple CMP or CMR fields from a query. (Deployment queries allow only a single element to be specified in the SELECT clause.) For more information, see the Example: EJB queries article.
- You want to return a computed expression in the query.
- You want to use value object methods or bean methods in the query statement. For more information, see Path expressions.
- You want to interactively test an EJB query during development, but do not want to repeatedly deploy your application each time you update a finder or select query.

The dynamic query API is a stateless session bean; using it is similar to using any other J2EE EJB application bean. It is included in the com.ibm.websphere.ejbquery in the API package.

The dynamic query bean has both a remote and a local interface. If you want to return remote EJB references from the query, or if the query statement contains remote methods, you must use the query remote interface:

```
remote interface = com.ibm.websphere.ejbquery.Query
remote home interface = com.ibm.websphere.ejbquery.QueryHome
```

If you want to return local EJB references from the query, or if the query statement contains local methods, you must use the query local interface:

```
local interface = com.ibm.websphere.ejbquery.QueryLocal
local home interface = com.ibm.websphere.ejbquery.QueryLocalHome
```

Because it uses less application server memory, the local interface ensures better overall EJB performance than the remote.

1. Verify that the query.ear application file is installed on the application server on which your application is to run, if that server is different from the default application server created during installation of the product.

   The `query.ear` file is located in the *app_server_root* directory, where `<WAS_HOME>` is the location of the WebSphere Application Server. The product installation program installs the `query.ear` file on the default application server using a JNDI name of

   ```
   com/ibm/websphere/ejbquery/Query
   ```

   (You or the system administrator can change this name.)
2. Set up authorization for the methods executeQuery(), prepareQuery(), and executePlan() in the remote and local dynamic query interfaces to control access to sensitive data. (This step is necessary only if your application requires security.)

   Because you cannot control which ASN names, CMP fields, or CMR fields can be used in a dynamic EJB query, you or your system administrator must place restrictions on use of the methods. If, for

example, a user is permitted to run the executeQuery method, he or she can run any valid dynamic query. In a production environment, you certainly want to restrict access to the remote query interface methods.

3. Write the dynamic query as part of your application client code. You can consult the following examples as query models; they illustrate which import statements to use, and so on:
   - Remote interface dynamic query example
   - Local interface dynamic query example
4. If the CMP you want to query is on a different module, you should:
   a. do a remote lookup on query.ear
   b. map the query.ear file to the server that the queried CMP bean is installed on.
5. Compile and run your client program with the file **qryclient.jar** in the classpath.

Example: Using the remote interface for Dynamic query

When you run a dynamic Enterprise JavaBeans (EJB) query using the remote interface, you are calling the executeQuery method on the Query interface. The executeQuery method has a transaction attribute of REQUIRED for this interface; therefore you do not need to explicitly establish a transaction context for the query to run.

Begin with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryHome;
import com.ibm.websphere.ejbquery.Query;
import com.ibm.websphere.ejbquery.QueryIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < 50000";
```

Create a Query object by obtaining a reference from the QueryHome class. (This class defines the executeQuery method.) Note that for the sake of simplicity, the following example uses the dynamic query JNDI name for the Query object:

```
InitialContext ic =  new InitialContext();

Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query");

QueryHome  qh =
 ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
Query qb = qh.create();
```

You then must specify a maximum size for the query result set, which is defined in the QueryIterator object, which is included in the Class QueryIterator. This class is included in the You then must specify a maximum size for the query result set, which is defined in the QueryIterator object, which is included in the QueryIterator API package. This example sets the maximum size of the result set to 99:

```
QueryIterator it = qb.executeQuery(query, null, null ,0, 99 );
```

The iterator contains a collection of IQueryTuple objects, which are records of the return collection values. Corresponding to the criteria of our example query statement, each tuple in this scenario contains one value of *name* and one value of *object(e)*. To display the contents of this query result, use the following code:

```
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
```

```
 System.out.println( it.getFieldName(2) );
 Emp e = ( Emp) javax.rmi.PortableRemoteObject.narrow( tuple.getObject(2), Emp.class );
 System.out.println( e.getPrimaryKey().toString());
}
```

The output from the program might look something like the following:

```
name Bob
emp 1001
name Dave
emp 298003
...
```

Finally, catch and process any exceptions. An exception might occur because of a syntax error in the query statement or a run-time processing error. The following example catches and processes these exceptions:

```
} catch (QueryException qe) {
    System.out.println("Query Exception "+ qe.getMessage() );
}
```

Handling large result collections for the remote interface query

If you intend your query to return a large collection, you have the option of programming it to return results in multiple smaller, more manageable quantities. Use the skipRow and maxRow parameters on the remote executeQuery method to retrieve the answer in chunks. For example:

```
int skipRow=0;
int maxRow=100;
QueryIterator it = null;
do {
 it = qb.executeQuery(query, null, null ,skipRow, maxRow );
 while (it.hasNext() ) {
 // display result
 skipRow = skipRow + maxRow;
}
} while ( ! it.isComplete() ) ;
```

Example: Using the local interface for Dynamic query

When you run a dynamic Enterprise JavaBeans (EJB) query using the local interface, you are calling the executeQuery method on the QueryLocal interface. This interface does not initiate a transaction for the method; therefore you must explicitly establish a transaction context for the query to run.

Note: To establish a transaction context, the following example calls the begin() and commit() methods. An alternative to using these methods is simply embedding your query code within an EJB method that runs within a transaction context.

Begin your query code with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryLocalHome;
import com.ibm.websphere.ejbquery.QueryLocal;
import com.ibm.websphere.ejbquery.QueryLocalIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name, object(e) from EmpBean e where e.salary < 50000 ";
```

Create a QueryLocal object by obtaining a reference from the QueryLocalHome class. (This class defines the executeQuery method.) Note that in the following example, `ejb/query` is used as a local EJB reference pointing to the dynamic query JNDI name (`com/ibm/websphere/ejbquery/Query`):

```
InitialContext ic =  new InitialContext();
   QueryLocalHome  qh =  ( LocalQueryHome) ic.lookup( "java:comp/env/ejb/query" );
QueryLocal qb = qh.create();
```

The last portion of code initiates a transaction, calls the executeQuery method, and displays the query results. The QueryLocalIterator class is instantiated because it defines the query result set. This class is included in the Class QueryIterator API package. Keep in mind that the iterator loses validity at the end of the transaction; you must use the iterator in the same transaction scope as the executeQuery call.

```
userTransaction.begin();
QueryLocalIterator it = qb.executeQuery(query, null, null);
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 EmpLocal e = ( EmpLocal ) tuple.getObject(2);
 System.out.println( e.getPrimaryKey().toString());
}
userTransaction.commit();
```

In most situations, the QueryLocalIterator object is *demand-driven*. That is, it causes data to be returned incrementally: for each record retrieval from the database, the next() method must be called on the iterator. (Situations can exist in which the iterator is not demand-driven. For more information, consult the ″Local query interfaces″ subsection of the Dynamic query performance considerations topic.)

Because the full query result set materializes incrementally in the application server memory, you can easily control its size. During a test run, for example, you may decide that return of only a few tuples of the query result is necessary. In that case you should use a call of the close() method on the QueryLocalIterator object to close the query loop. Doing so frees SQL resources that the iterator uses. Otherwise, these resources are not freed until the full result set accumulates in memory, or the transaction ends.

Dynamic query performance considerations

While using a dynamic query can be convenient, there are times when it can have an impact on your application performance.

General performance considerations

Use of the following elements in your dynamic query can diminish application performance somewhat:

- Datatype converters and Java methods

  Why: In general, query operations and predicates are translated into SQL so that the database server can perform them. If your query includes datatype converters (for EJB to RDB mapping, for example) or Java methods, however, the associated predicates and operations of your query must be performed in the memory of the application server.
- EJB methods and criteria that call for the return of EJB references

  Why: Queries that incorporate these elements trigger full activation of EJBs in the memory of the application server. (Returning a list of CMP fields from a query does not cause an EJB to be activated.)

When assessing application performance, you should also be aware that dynamic queries share connections with the persistence manager. Consequently, an application that includes a mixture of finder methods, CMR navigation, and dynamic queries relies on a single shared connection between the persistence manager and the dynamic query service to perform these tasks.

Limiting the return collection size

- **Remote interface queries**: The QueryIterator class of the remote interface mandates that all of your query results materialize in application server memory over the course of one method call. The SQL

cursor(s) used to run the EJB query are closed upon completion of that call. Because this requirement poses a high risk for creating bottlenecks within the database server, you need to limit the size of any potentially large result collections.

- **Local interface queries**: In most situations, the QueryLocalIterator object behaves as a wrapper around an SQL cursor. It is *demand-driven*; it causes data to be returned incrementally. For each record retrieval from the database, the next() method must be called on the iterator.

  Use of certain operations in local interface queries, however, overrides the demand-driven behavior. In these cases, the query results fully materialize in memory just as do the result collections of remote interface queries. An example of such a case is:

  ```
  select  e.myBusinessMethod( ) from EmpBean e
   where e.salary < 50000 order by 1 desc
  ```

  This query requires performance of an EJB method to produce the final result collection. Consequently, the full dataset from the database must be returned in one collection to application server memory, where the EJB method can be run on the dataset in its entirety. For that reason, local interface query operations that invoke EJB methods are generally not demand-driven. You cannot control the return collection size for such queries.

  Because they *are* demand-driven, all other local interface queries allow you to control the size of return collections. You can use a call of the close() method on the QueryLocalIterator object to close the query loop after the desired number of return values has been fetched from the datastore. Otherwise, the SQL cursor(s) used to run the EJB query are not closed until the full result set accumulates in memory, or the transaction ends.

Access intent implications for dynamic query

WebSphere Application Server gives you the option to set access intent policies for your entity enterprise beans as a way of managing their transfer of data with the underlying data store. An access intent policy controls the isolation level used on the data source connection, as well as the database locks used during data retrieval. By manipulating these elements, you can maximize the efficiency of your application's data flow.

To learn more, begin with the topics "Access intent policies" on page 161 and "Concurrency control" on page 161.

When formulating dynamic queries, keep in mind the following considerations concerning their interaction with access intent policies:

- A dynamic query uses the first ASN name in the FROM clause to determine access intent.
- The collection increment attribute of an access intent policy is not used in processing a dynamic query.
- When performed on entity beans that have a pessimistic-Update access intent policy, your dynamic queries must return updateable collections. Therefore you need to formulate your query statements to return only collections of entity beans, *not* collections of CMP fields. For example, the statement `select object(c) from Customer` is valid for a dynamic query performed under the constraint of a pessimistic-Update policy. The statement `select c.name from Customer c`, however, is not a valid dynamic query under this constraint.
- Using pessimistic-Update policy places restrictions on the types of query expressions. The restrictions depend on the back end database type and release. Refer to the topic Access intent -- isolation levels and update locks for details.

Dynamic query API: prepareQuery() and executePlan() methods

Use these methods to more efficiently allocate the overhead associated with dynamic query. They are equivalent in function to the prepareStatement() and executeQuery() methods of the JDBC API.

To perform a dynamic Enterprise JavaBeans (EJB) query, the application server must parse the query string into structured query language (SQL) at run time. You can, of course, eliminate run-time overhead

by choosing to perform a standard EJB query instead of a dynamic query. Sometimes referred to as *deployment queries*, standard queries are parsed and built at deployment, then performed by a finder or select method.

Another option is to write code that redistributes dynamic query overhead for better application performance. Begin by calling the prepareQuery() method in place of the executeQuery() method. The prepareQuery() method parses and translates your query, and returns a string called a *query plan*. The plan contains the SQL statement produced by parsing and translation, as well as other information needed by the dynamic query API. Save this string in your application and call the executePlan() method with the string to run your query. (You also might want to use the prepareQuery() method simply to see the SQL translation product; just call the method and display the return value.)

Pass the parameters of your query as an array of type Object on the prepareQuery() and the executePlan() method calls. Ensure that you pass appropriate data types, because the application server validates your query according to parameter type (rather than actual values) when it processes the prepareQuery() method call.

Example code

Note: In the example code that follows, the first executePlan() method call substitutes `parms[0]` for `?1`. Hence the first query performed is functionally equivalent to the following query statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

The second call runs a query that is functionally equivalent to this statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 60000
```

The example:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < ?1";
QueryIterator it = null;
Integer[] parms = new Integer[1];
parms[0] = new Integer(0);
```

In the call to `prepareQuery()`, pass any Integer value. Doing so defines `?1` as an Integer type, as in the following:

```
String queryPlan= qb.prepareQuery(query, parms, null );

   parms[0] = new Integer(50000);
```

Next you run the query with a real value of `Integer(50000)` for `?1`:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000it =
 qb.executePlan( queryPlan, parms, 0, 99);

parms[0] = new Integer(60000);
```

Run the query again with a different value of `Integer(60000)` for `?1`:

```
it = qb.executePlan( queryPlan, parms, 0, 99);
```

Related tasks

“Using the dynamic query service” on page 2310
There are times in the development process when you might prefer to use the dynamic query service rather than the regular Enterprise JavaBean (EJB) query service (which can be referred to as *deployment query*). During testing, for instance, the dynamic query service can be used at application run time, so you do not have to re-deploy your application.

Using the dynamic query service

There are times in the development process when you might prefer to use the dynamic query service rather than the regular Enterprise JavaBean (EJB) query service (which can be referred to as *deployment query*). During testing, for instance, the dynamic query service can be used at application run time, so you do not have to re-deploy your application.

About this task

Following are common reasons for using the dynamic query service rather than the regular EJB query service:

- You need to programmatically define a query at application run time, rather than at deployment.
- You need to return multiple CMP or CMR fields from a query. (Deployment queries allow only a single element to be specified in the SELECT clause.) For more information, see the Example: EJB queries article.
- You want to return a computed expression in the query.
- You want to use value object methods or bean methods in the query statement. For more information, see Path expressions.
- You want to interactively test an EJB query during development, but do not want to repeatedly deploy your application each time you update a finder or select query.

The dynamic query API is a stateless session bean; using it is similar to using any other J2EE EJB application bean. It is included in the com.ibm.websphere.ejbquery in the API package.

The dynamic query bean has both a remote and a local interface. If you want to return remote EJB references from the query, or if the query statement contains remote methods, you must use the query remote interface:

remote interface = `com.ibm.websphere.ejbquery.Query`
remote home interface = `com.ibm.websphere.ejbquery.QueryHome`

If you want to return local EJB references from the query, or if the query statement contains local methods, you must use the query local interface:

local interface = `com.ibm.websphere.ejbquery.QueryLocal`
local home interface = `com.ibm.websphere.ejbquery.QueryLocalHome`

Because it uses less application server memory, the local interface ensures better overall EJB performance than the remote.

1. Verify that the query.ear application file is installed on the application server on which your application is to run, if that server is different from the default application server created during installation of the product.

   The `query.ear` file is located in the *app_server_root* directory, where `<WAS_HOME>` is the location of the WebSphere Application Server. The product installation program installs the `query.ear` file on the default application server using a JNDI name of

   `com/ibm/websphere/ejbquery/Query`

   (You or the system administrator can change this name.)

2. Set up authorization for the methods executeQuery(), prepareQuery(), and executePlan() in the remote and local dynamic query interfaces to control access to sensitive data. (This step is necessary only if your application requires security.)

   Because you cannot control which ASN names, CMP fields, or CMR fields can be used in a dynamic EJB query, you or your system administrator must place restrictions on use of the methods. If, for example, a user is permitted to run the executeQuery method, he or she can run any valid dynamic query. In a production environment, you certainly want to restrict access to the remote query interface methods.
3. Write the dynamic query as part of your application client code. You can consult the following examples as query models; they illustrate which import statements to use, and so on:
   - Remote interface dynamic query example
   - Local interface dynamic query example
4. If the CMP you want to query is on a different module, you should:
   a. do a remote lookup on query.ear
   b. map the query.ear file to the server that the queried CMP bean is installed on.
5. Compile and run your client program with the file **qryclient.jar** in the classpath.

Example: Using the remote interface for Dynamic query

When you run a dynamic Enterprise JavaBeans (EJB) query using the remote interface, you are calling the executeQuery method on the Query interface. The executeQuery method has a transaction attribute of REQUIRED for this interface; therefore you do not need to explicitly establish a transaction context for the query to run.

Begin with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryHome;
import com.ibm.websphere.ejbquery.Query;
import com.ibm.websphere.ejbquery.QueryIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < 50000";
```

Create a Query object by obtaining a reference from the QueryHome class. (This class defines the executeQuery method.) Note that for the sake of simplicity, the following example uses the dynamic query JNDI name for the Query object:

```
InitialContext ic =  new InitialContext();

Object obj =  ic.lookup("com/ibm/websphere/ejbquery/Query");

QueryHome  qh =
 ( QueryHome) javax.rmi.PortableRemoteObject.narrow( obj, QueryHome.class );
Query qb = qh.create();
```

You then must specify a maximum size for the query result set, which is defined in the QueryIterator object, which is included in the Class QueryIterator. This class is included in the You then must specify a maximum size for the query result set, which is defined in the QueryIterator object, which is included in the QueryIterator API package. This example sets the maximum size of the result set to 99:

```
QueryIterator it = qb.executeQuery(query, null, null ,0, 99 );
```

The iterator contains a collection of IQueryTuple objects, which are records of the return collection values. Corresponding to the criteria of our example query statement, each tuple in this scenario contains one value of *name* and one value of *object(e)*. To display the contents of this query result, use the following code:

```
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 Emp e = ( Emp) javax.rmi.PortableRemoteObject.narrow( tuple.getObject(2), Emp.class );
 System.out.println( e.getPrimaryKey().toString());
}
```

The output from the program might look something like the following:

```
name Bob
emp 1001
name Dave
emp 298003
...
```

Finally, catch and process any exceptions. An exception might occur because of a syntax error in the query statement or a run-time processing error. The following example catches and processes these exceptions:

```
} catch (QueryException qe) {
    System.out.println("Query Exception "+ qe.getMessage() );
}
```

Handling large result collections for the remote interface query

If you intend your query to return a large collection, you have the option of programming it to return results in multiple smaller, more manageable quantities. Use the skipRow and maxRow parameters on the remote executeQuery method to retrieve the answer in chunks. For example:

```
int skipRow=0;
int maxRow=100;
QueryIterator it = null;
do {
 it = qb.executeQuery(query, null, null ,skipRow, maxRow );
 while (it.hasNext() ) {
 // display result
 skipRow = skipRow + maxRow;
}
} while ( ! it.isComplete() ) ;
```

Example: Using the local interface for Dynamic query

When you run a dynamic Enterprise JavaBeans (EJB) query using the local interface, you are calling the executeQuery method on the QueryLocal interface. This interface does not initiate a transaction for the method; therefore you must explicitly establish a transaction context for the query to run.

Note: To establish a transaction context, the following example calls the begin() and commit() methods. An alternative to using these methods is simply embedding your query code within an EJB method that runs within a transaction context.

Begin your query code with the following import statements:

```
import com.ibm.websphere.ejbquery.QueryLocalHome;
import com.ibm.websphere.ejbquery.QueryLocal;
import com.ibm.websphere.ejbquery.QueryLocalIterator;
import com.ibm.websphere.ejbquery.IQueryTuple;
import com.ibm.websphere.ejbquery.QueryException;
```

Next, write your query statement in the form of a string, as in the following example that retrieves the names and ejb-references for underpaid employees:

```
String query =
"select e.name, object(e) from EmpBean e where e.salary < 50000 ";
```

Create a QueryLocal object by obtaining a reference from the QueryLocalHome class. (This class defines the executeQuery method.) Note that in the following example, `ejb/query` is used as a local EJB reference pointing to the dynamic query JNDI name (`com/ibm/websphere/ejbquery/Query`):

```
InitialContext ic =  new InitialContext();
   QueryLocalHome  qh =  ( LocalQueryHome) ic.lookup( "java:comp/env/ejb/query" );
QueryLocal qb = qh.create();
```

The last portion of code initiates a transaction, calls the executeQuery method, and displays the query results. The QueryLocalIterator class is instantiated because it defines the query result set. This class is included in the Class QueryIterator API package. Keep in mind that the iterator loses validity at the end of the transaction; you must use the iterator in the same transaction scope as the executeQuery call.

```
userTransaction.begin();
QueryLocalIterator it = qb.executeQuery(query, null, null);
while (it.hasNext() ) {
 IQueryTuple tuple = (IQueryTuple) it.next();
 System.out.print( it.getFieldName(1) );
 String s = (String) tuple.getObject(1);
 System.out.println( s);
 System.out.println( it.getFieldName(2) );
 EmpLocal e = ( EmpLocal ) tuple.getObject(2);
 System.out.println( e.getPrimaryKey().toString());
}
userTransaction.commit();
```

In most situations, the QueryLocalIterator object is *demand-driven*. That is, it causes data to be returned incrementally: for each record retrieval from the database, the next() method must be called on the iterator. (Situations can exist in which the iterator is not demand-driven. For more information, consult the ″Local query interfaces″ subsection of the Dynamic query performance considerations topic.)

Because the full query result set materializes incrementally in the application server memory, you can easily control its size. During a test run, for example, you may decide that return of only a few tuples of the query result is necessary. In that case you should use a call of the close() method on the QueryLocalIterator object to close the query loop. Doing so frees SQL resources that the iterator uses. Otherwise, these resources are not freed until the full result set accumulates in memory, or the transaction ends.

Dynamic query performance considerations

While using a dynamic query can be convenient, there are times when it can have an impact on your application performance.

General performance considerations

Use of the following elements in your dynamic query can diminish application performance somewhat:

- Datatype converters and Java methods

  Why: In general, query operations and predicates are translated into SQL so that the database server can perform them. If your query includes datatype converters (for EJB to RDB mapping, for example) or Java methods, however, the associated predicates and operations of your query must be performed in the memory of the application server.
- EJB methods and criteria that call for the return of EJB references

  Why: Queries that incorporate these elements trigger full activation of EJBs in the memory of the application server. (Returning a list of CMP fields from a query does not cause an EJB to be activated.)

When assessing application performance, you should also be aware that dynamic queries share connections with the persistence manager. Consequently, an application that includes a mixture of finder methods, CMR navigation, and dynamic queries relies on a single shared connection between the persistence manager and the dynamic query service to perform these tasks.

Limiting the return collection size

- **Remote interface queries**: The QueryIterator class of the remote interface mandates that all of your query results materialize in application server memory over the course of one method call. The SQL cursor(s) used to run the EJB query are closed upon completion of that call. Because this requirement poses a high risk for creating bottlenecks within the database server, you need to limit the size of any potentially large result collections.
- **Local interface queries**: In most situations, the QueryLocalIterator object behaves as a wrapper around an SQL cursor. It is *demand-driven*; it causes data to be returned incrementally. For each record retrieval from the database, the next() method must be called on the iterator.

  Use of certain operations in local interface queries, however, overrides the demand-driven behavior. In these cases, the query results fully materialize in memory just as do the result collections of remote interface queries. An example of such a case is:

  ```
  select  e.myBusinessMethod( ) from EmpBean e
   where e.salary < 50000 order by 1 desc
  ```

  This query requires performance of an EJB method to produce the final result collection. Consequently, the full dataset from the database must be returned in one collection to application server memory, where the EJB method can be run on the dataset in its entirety. For that reason, local interface query operations that invoke EJB methods are generally not demand-driven. You cannot control the return collection size for such queries.

  Because they *are* demand-driven, all other local interface queries allow you to control the size of return collections. You can use a call of the close() method on the QueryLocalIterator object to close the query loop after the desired number of return values has been fetched from the datastore. Otherwise, the SQL cursor(s) used to run the EJB query are not closed until the full result set accumulates in memory, or the transaction ends.

Access intent implications for dynamic query

WebSphere Application Server gives you the option to set access intent policies for your entity enterprise beans as a way of managing their transfer of data with the underlying data store. An access intent policy controls the isolation level used on the data source connection, as well as the database locks used during data retrieval. By manipulating these elements, you can maximize the efficiency of your application's data flow.

To learn more, begin with the topics "Access intent policies" on page 161 and "Concurrency control" on page 161.

When formulating dynamic queries, keep in mind the following considerations concerning their interaction with access intent policies:

- A dynamic query uses the first ASN name in the FROM clause to determine access intent.
- The collection increment attribute of an access intent policy is not used in processing a dynamic query.
- When performed on entity beans that have a pessimistic-Update access intent policy, your dynamic queries must return updateable collections. Therefore you need to formulate your query statements to return only collections of entity beans, *not* collections of CMP fields. For example, the statement `select object(c) from Customer` is valid for a dynamic query performed under the constraint of a pessimistic-Update policy. The statement `select c.name from Customer c`, however, is not a valid dynamic query under this constraint.
- Using pessimistic-Update policy places restrictions on the types of query expressions. The restrictions depend on the back end database type and release. Refer to the topic Access intent -- isolation levels and update locks for details.

Dynamic query API: prepareQuery() and executePlan() methods

Use these methods to more efficiently allocate the overhead associated with dynamic query. They are equivalent in function to the prepareStatement() and executeQuery() methods of the JDBC API.

To perform a dynamic Enterprise JavaBeans (EJB) query, the application server must parse the query string into structured query language (SQL) at run time. You can, of course, eliminate run-time overhead by choosing to perform a standard EJB query instead of a dynamic query. Sometimes referred to as *deployment queries*, standard queries are parsed and built at deployment, then performed by a finder or select method.

Another option is to write code that redistributes dynamic query overhead for better application performance. Begin by calling the prepareQuery() method in place of the executeQuery() method. The prepareQuery() method parses and translates your query, and returns a string called a *query plan*. The plan contains the SQL statement produced by parsing and translation, as well as other information needed by the dynamic query API. Save this string in your application and call the executePlan() method with the string to run your query. (You also might want to use the prepareQuery() method simply to see the SQL translation product; just call the method and display the return value.)

Pass the parameters of your query as an array of type Object on the prepareQuery() and the executePlan() method calls. Ensure that you pass appropriate data types, because the application server validates your query according to parameter type (rather than actual values) when it processes the prepareQuery() method call.

Example code

Note: In the example code that follows, the first executePlan() method call substitutes `parms[0]` for `?1`. Hence the first query performed is functionally equivalent to the following query statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000
```

The second call runs a query that is functionally equivalent to this statement:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 60000
```

The example:

```
String query =
"select e.name as name , object(e) as emp from EmpBean e where e.salary < ?1";
QueryIterator it = null;
Integer[] parms = new Integer[1];
parms[0] = new Integer(0);
```

In the call to `prepareQuery()`, pass any Integer value. Doing so defines `?1` as an Integer type, as in the following:

```
String queryPlan= qb.prepareQuery(query, parms, null );

   parms[0] = new Integer(50000);
```

Next you run the query with a real value of `Integer(50000)` for `?1`:

```
select e.name as name, object(e) as emp from EmpBean e where e.salary < 50000it =
 qb.executePlan( queryPlan, parms, 0, 99);

parms[0] = new Integer(60000);
```

Run the query again with a different value of `Integer(60000)` for `?1`:

```
it = qb.executePlan( queryPlan, parms, 0, 99);
```

Related tasks

"Using the dynamic query service" on page 2310
There are times in the development process when you might prefer to use the dynamic query service rather than the regular Enterprise JavaBean (EJB) query service (which can be referred to as *deployment query*). During testing, for instance, the dynamic query service can be used at application run time, so you do not have to re-deploy your application.

Internationalization

Task overview: Globalizing applications

An application that can present information to users according to regional cultural conventions is said to be *globalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In a globalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region. Globalization consists of two phases: *internationalization* (enabling an application component to use regional conventions) and *localization* (implementing a specific regional convention). This product supports globalization through the use of its localizable-text API and internationalization service.

- Make sure the server runtime environment is properly configured.

  For more information about supported locales and character encodings, see "Working with locales and character encodings" on page 2328.
- Implement message catalogs in your application by using the localizable-text API.

  This product supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

  For more information about the localizable-text API, see "Task overview: Internationalizing interface strings (localizable-text API)" on page 2325.
- Implement more extensive locale support by using the internationalization service.

  With the internationalization service, you can manage the distribution of the internationalization information, or *internationalization context*, that is necessary to perform localizations within Java Platform, Enterprise Edition (Java EE) application components. Supported application components also include Web service client environments and Web service-enabled enterprise beans.

  For more information about the internationalization service, see "Task overview: Internationalizing application components (internationalization service)" on page 2326.

Globalization

An application that can present information to users according to regional cultural conventions is said to be *globalized*: The application can be configured to interact with users from different localities in culturally appropriate ways. In a globalized application, a user in one region sees error messages, output, and interface elements in the requested language. Date and time formats, as well as currencies, are presented appropriately for users in the specified region. A user in another region sees output in the conventional language or format for that region. Globalization consists of two phases: *internationalization* (enabling an application component to use regional conventions) and *localization* (implementing a specific regional convention).

Historically, the creation of globalized applications has been restricted to large corporations writing complex systems. However, given the rise in distributed computing and in the use of the World Wide Web, application developers are pressured to globalize a much wider variety of applications. This trend requires making globalization techniques much more accessible to application developers.

Internationalization of an application is driven by two variables, the time zone and the locale. The *time zone* indicates how to compute the local time as an offset from a standard time like Greenwich Mean Time. The *locale* is a collection of information about language, currency, and the conventions for

presenting information like dates. A time zone can cover many locales, and a single locale can span time zones. With both time zone and locale, the date, time, currency, and language for users in a specific region can be determined.

By convention, a given locale is specified with a pair of codes (for language and region) that are governed by different standards. The ISO-639 standard governs the language code; the ISO-3166 standard governs the regional code. In notation, the two codes are typically joined by an underscore (_) character, for example, en_US for English in the United States. In Java code, locales are set and retrieved by means of the java.util.Locale class.

A first step: Localization of interface strings

In an application that is not globalized, the user interface is unalterably written into the application code. Internationalizing a user interface adds a layer of abstraction into the design of an application. The additional layer of abstraction enables you to localize the application for each locale that must be supported by the application.

In a localized application, the locale determines the message catalog from which the application retrieves message strings. Instead of printing an error message, the application represents the error message with some language-neutral information; in the simplest case, each error condition corresponds to a key. To print a usable error message, the application looks up the key in a *message catalog*. Each message catalog is a list of keys with associated strings. Different message catalogs provide strings for the different languages that are supported. The application looks up the key in the appropriate catalog, retrieves the corresponding error message in the requested language, and prints the string for the user.

Localization of text can be used for far more than translating error messages. For example, by using keys to represent each element in a graphical user interface (GUI) and by providing the appropriate message catalogs, the GUI (buttons, menus, and so on) can support multiple languages. Extending support to additional languages requires that you provide message catalogs for those languages; in many cases, the application needs no further modification.

The localizable-text package is a set of Java classes and interfaces that can be used to localize the strings in distributed applications easily. Language-specific string catalogs can be stored centrally so that they can be maintained efficiently.

Globalization challenges in distributed applications

With the advent of Internet-based business computational models, applications increasingly consist of clients and servers that operate in different geographical regions. These differences introduce the following challenges to the task of designing a solid client-server infrastructure:

Clients and servers can run on computers that have different endian architectures or code sets

Clients and servers can reside in computers that have different endian architectures: A client can reside in a little-endian CPU, while the server code runs in a big-endian one. A client might want to call a business method on a server running in a code set different from that of the client.

A client-server infrastructure must define precise endian and code-set tracking and conversion rules. The Java platform has nearly eliminated these problems in a unique way by relying on its Java virtual machine (JVM), which encodes all of the string data in UCS-2 format and externalizes everything in big-endian format. The JVM uses a set of platform-specific programs for interfacing with the native platform. These programs perform any necessary code set conversions between UCS-2 and the native code set of a platform.

Clients and servers can run on computers with different locale settings

Client and server processes can use different locale settings. For example, a Spanish client might call a business method upon an object that resides on an American English server. Some

business methods are locale-sensitive in nature; for example, given a business method that returns a sorted list of strings, the Spanish client expects that list to be sorted according to the Spanish collating sequence, not in the English collating sequence of the server. Because data retrieval and sorting procedures run on the server, the locale of the client must be available to perform a legitimate sort.

A similar consideration applies in instances where the server has to return strings containing date, time, currency, exception messages, and so on, that are formatted according to the cultural expectations of the client.

Clients and servers can reside in different time zones

Client and server processes can run in different time zones. To date, all internationalization literature and resources concentrate mainly on code set and locale-related issues. They have generally ignored the time zone issue, even though business methods can be sensitive to time zone as well as to locale.

For example, suppose that a vendor makes the claim that orders received before 2:00 PM are processed by 5:00 PM the same day. The times given, of course, are in the time zone of the server that is processing the order. It is important to know the time zone of the client to give customers in other time zones the correct times for same-day processing.

Other time zone-sensitive operations include time stamping messages logged to a server, and accessing file or database resources. The concept of Daylight Savings Time further complicates the time zone issue.

Java Platform, Enterprise Edition (Java EE) provides support for application components that run on computers with differing endian architecture and code sets. It does not provide dedicated support for application components that run on computers with different locales or time zones.

The conventional method for solving locale and time zone mismatches across remote application components is to pass one or more extra parameters on all business methods needed to convey the client-side locale or time zone to the server. Although simple, this technique has the following limitations when used in Enterprise JavaBeans (EJB) applications:

- It is intrusive because it requires that one or more parameters be added to all bean methods in the call chain to locale-sensitive or time zone-sensitive methods.
- It is inherently error-prone.
- It is impracticable within applications that do not support modification, such as legacy applications.

The internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. The service systematically manages the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets. For more information, see “Task overview: Internationalizing application components (internationalization service)” on page 2326.

Language versions offered by this product

This product is offered in several languages, as enabled by the operating platform on which the product is installed.

The following language versions are available:

- Brazilian Portuguese
- Chinese (Simplified)
- Chinese (Traditional)
- Czech
- English
- French
- German

- Hungarian
- Italian
- Japanese
- Korean
- Polish
- Russian
- Spanish

Globalization: Resources for learning

Use links in this topic to find relevant supplemental information about globalization. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to this product but is useful all or in part for understanding the product. When possible, links are provided to technical papers and IBM Redbooks publications that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

View links to additional information about:
- “Programming instructions and examples”
- “Programming specifications”

Programming instructions and examples

- Java internationalization tutorial

  An online tutorial that explains how to use the Java SDK Internationalization API.
- Globalize your On Demand Business

  IBM’s portal site for delivering globalized applications.

Programming specifications

- Java 2 Platform Standard Edition 5.0 Development Kit Documentation: Internationalization

  The Java internationalization documentation from Sun Microsystems, including a list of supported locales and encodings. For other versions of the Java platform, click the ″Internationalization Home Page″ link on that page.
- Java Specification Request 150, Internationalization Service for J2EE

  The specification of the Java internationalization service that was developed through the Java Community Process.
- W3C, Internationalization Core Working Group

  The W3C’s Internationalization Core Working Group responsible for investigating the internationalization of Web services, in particular, the dependence of Web services on language, culture, region, and locale-related contexts.
- Making the WWW truly World Wide

  The W3C effort to make World Wide Web technology work with the many writing systems, languages, and cultural conventions of the global community:

Task overview: Internationalizing interface strings (localizable-text API)

This topic summarizes the steps involved in implementing message catalogs through the localizable-text API.

About this task

This product supports the maintenance and deployment of centralized message catalogs for the output of properly formatted, language-specific (*localized*) interface strings.

1. Identify localizable text in your application.

2. Create the message catalogs that are necessary for the locales to be supported by your application.
3. In your application code, compose the language-specific strings for output.
4. Using an assembly tool, assemble your application code as one or more application components.
5. Prepare the localizable-text package for deployment with your localized application. In this step, you create a deployment Java archive (JAR) file.
6. Assemble the application modules and the deployment JAR file into a Java Platform, Enterprise Edition (Java EE) application.
7. Deploy and manage the application.

Results

Your application is deployed with localized text.

Task overview: Internationalizing application components (internationalization service)

This topic summarizes the steps involved in using the internationalization service.

About this task

With the internationalization service, you can manage the distribution of the internationalization information, or *internationalization context*, that is necessary to perform localizations within Java Platform, Enterprise Edition (Java EE) application components. Supported application components also include Web service client environments and Web service-enabled enterprise beans.

1. Use the internationalization context API within application components to obtain or manage internationalization context.

   Servlet and enterprise bean business methods can use internationalization context to perform locale- and time zone-sensitive localizations. Enterprise JavaBeans (EJB) client applications, and server components that are configured to manage internationalization context must use the internationalization context API to set the context elements scoped to their invocations.

   You use the internationalization context API within Web service-enabled Java EE client programs and stateless session beans in the same manner that you would use conventional Java EE application components, with one exception. Internationalization context propagated over Web service requests contains a time zone ID, whereas conventional Remote Method Invocation/ Internet Inter-ORB Protocol (RMI/IIOP) requests propagate complete time zone information, including the raw offset, Daylight Savings Time information, and so on.
2. Assemble internationalized applications.

   The internationalization type specifies the internationalization policy that applies to a servlet or an enterprise bean and, in particular, indicates whether the application component or its hosting Java EE container manages internationalization context. Container internationalization attributes can be specified for container-managed servlet and enterprise bean business methods. These attributes tailor a policy by indicating which context the container scopes to an invocation. Configuring internationalization policies declaratively prescribes, by means of the application deployment descriptor, the distribution and management of context throughout an application.

   As you edit the deployment descriptor for assembly, you can also set the internationalization type and configure any container internationalization attributes for the servlets and enterprise beans in your application.

   You configure internationalization type and container internationalization attributes for Web service-enabled stateless session beans in the same manner as you do for conventional beans.
3. Manage the internationalization service.

   Use the administrative console to enable the service on all application servers.

By default, the service is enabled within Java EE client environments but is disabled on application servers. You must enable the service on all application servers hosting your servlets and enterprise beans to use internationalization context.

4. Troubleshoot the internationalization service as needed.

   Use the administrative console to enable the trace service to log internationalization service messages when debugging your applications.

   The trace strings for the internationalization service follow; use both:

   ```
   com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled
   ```

Internationalization service

In a distributed client-server environment, application processes can run on different machines, configured for different locales, corresponding to different cultural conventions; they can also be located across geographical boundaries. The internationalization service can help manage your application in a globally distributed environment.

For an understanding of how differences in locale impact application development, read “Globalization” on page 2322.

Java Platform, Enterprise Edition (Java EE) provides support for application components that run on computers with differing endian architecture and code sets. It does not provide dedicated support for application components that run on computers with different locales or time zones.

The internationalization service addresses the challenges posed by locale and time zone mismatch without incurring the limitations of conventional techniques. The service systematically manages the distribution of internationalization contexts across the various components of EJB applications, including client applications, enterprise beans, and servlets.

The service works by associating an internationalization context with every service request within an application. When a client-side component calls a business method, the internationalization service interposes by obtaining the internationalization context associated with the current client-side process and by attaching that context to the outgoing request. On the server side, the internationalization service again interposes by detaching the context from the incoming request and associating it with the server-side process on which the business method will run, effectively scoping the context to the business method. For HTTP requests, the caller context is constructed from the HTTP attributes and default values. The service propagates internationalization context on subsequent business method invocations in the same manner, which distributes the context of the originating request over the entire chain of business method invocations.

This basic operation of scoping and propagation is defined precisely by *internationalization context management policies*. Internationalization policies specify whether an application component or its hosting Java EE container are to manage internationalization context. For container-managed components, the policy indicates which internationalization context the container scopes to invocations on that component. Server components configured to manage internationalization context, as well as EJB clients, must use the internationalization context API to manage the internationalization context elements scoped to their invocations.

Every application component has a default policy, which can be overridden and tailored for servlets and enterprise beans at assembly time.

At run time, application components can use the internationalization context API to get any element of the internationalization contexts scoped to an invocation. To programmatically access context elements, application components first resolve an internationalization context API reference, then call the appropriate API method to access the various context elements, such as the caller locale or the invocation time zone. These elements can be used in calls to Java SDK internationalization API methods; for example, to perform localizations such as formatting messages, configuring dates, or comparing strings.

Working with locales and character encodings

Internationalization support for this product relies on that provided by the Java Platform, Standard Edition (JSE). Support varies by platform.

- Verify that the operating system on which the application server is installed supports the locales and encodings that you plan to use.

  Java internationalization support might use underlying services of the operating system. For example, if user IDs for your server are expected to contain non-English characters, make sure that the operating system is configured to process those characters.
- Plan for encoding changes as necessary.

  Consider differences in encoding support among operating system subcomponents. Although this product and the Java platform are based on Unicode encoding, it is not always possible to run applications in a purely Unicode environment.
- Set the console.encoding property as necessary.

Results

If your application produces an UnsupportedEncodingException exception, check your operating system documentation to determine if the target operating system supports the required encoding and adjust the runtime environment as needed.

Administering the internationalization service

To use internationalization context in an Enterprise JavaBeans (EJB) application, the internationalization service must be enabled in the runtime environments for all server-side components (servlets and enterprise beans, including session beans enabled for Web service usage) as well as all client-side components (EJB client applications and Web service clients).

About this task

If you do not require the internationalization service, do not enable it. Leaving the service disabled prevents any possible performance degradation incurred by the implicit distribution of internationalization resources.

The internationalization service cannot be enabled for HTTP clients, because support for internationalization in that case is provided by the browser, not by the application server.

- Enable or disable the internationalization service for servlets and enterprise beans. By default, the service is disabled for server-side components within the application server. You enable the service by using either the administrative console or the wsadmin tool.
- Enable or disable the internationalization service for EJB clients. By default, the service is disabled within the client container. You enable the service by using the launchClient tool.

Enabling the internationalization service for servlets and enterprise beans

Perform this task to enable the internationalization service in the application server runtime environment.

About this task

Any servlet or enterprise bean can use internationalization context if the internationalization service is enabled within the hosting application server instance.

1. Start the administrative console.
2. Click **Servers** > **Application servers** > *server_name* > **Container services** > **Internationalization service**.
3. Enable the internationalization service.
   a. If not already selected, select the **Enable service at server startup** check box.

b. Click **OK**.

Results

When you select the **Enable service at server startup** setting, the application server automatically initializes and starts the internationalization service whenever the server starts. If you change this setting, be sure to restart the application server for the new setting to take effect.

To disable the service, clear the **Enable service at server startup** check box. In this case, the internationalization service is initialized but not started when the application server starts.

Example

Alternatively, the internationalization service can be enabled from the command line by using the wsadmin tool. Start the wsadmin tool and enter the following commands:

```
set x [$AdminConfig list I18NService]
$AdminConfig modify $x { { enable true } }
$AdminConfig save
exit
```

What to do next

If you enable or disable the internationalization service, be sure to stop and then restart the application server for the new setting to take effect.

Enabling the internationalization service for EJB clients

By default, the internationalization service is disabled for use within Enterprise JavaBeans (EJB) and Web-service enabled client applications. You must enable the service for client applications as well as for all server instances in the runtime environment.

Enable the service.
When calling the launchClient tool, include the argument `-CCDI18NService.enable=true` or `-CCDI18NService.enable=yes`.

Internationalization service settings

Use this page to enable or disable the internationalization service. The internationalization service manages the implicit propagation and scoping of locale and time zone information, called *internationalization context*, within application components. When the service is enabled, application components can use the internationalization context API to programmatically manage locale and time zone information. In turn, components can use that locale and time zone information with the Java Platform, Standard Edition (JSE) Internationalization API to perform localizations. If internationalization support is not required on the server, disabling the service can improve performance.

To view this administrative console page, click **Servers > Server Types > WebSphere application servers >** *server_name*. Then, under Container Settings, click **Container Services > Internationalization Service**.

Enable service at server startup:

Specifies whether the server attempts to start the internationalization service.

Default	Cleared
Range	Valid values are Selected or Cleared

More information about valid values follows:

Selected
When the application server starts, it attempts to start the internationalization service automatically.
Cleared
The server does not try to start the internationalization service.

To enable the internationalization service for applications on this server, the system administrator must select this property and then restart the server.

Internationalization service errors

Certain conditions might cause the internationalization service not to start, to issue java.lang.IllegalStateException exceptions while an application is running, or to exercise default behaviors.

The java.lang.IllegalStateException exception indicates one of the following things:

- An application component attempted an operation that is not supported by the internationalization programming model.

  The IllegalStateException exception is issued whenever a server application component whose internationalization type is set to container-managed internationalization (CMI) attempts to set invocation context. This behavior is a violation of the CMI policy, under which servlets and enterprise beans cannot modify their invocation internationalization context.
- An anomaly occurred that disabled the service.

  For instance, if the internationalization service is not properly initialized, the Java Naming and Directory Interface (JNDI) lookup on the UserInternationalization URL attribute issues a javax.naming.NameNotFoundException exception that contains an IllegalStateException instance.

The following conditions can occur while your internationalized application is running. These conditions might cause the internationalization service not to start, to issue IllegalStateException exceptions, or to exercise default behaviors:

- “The service is disabled ”
- “The service is not started” on page 2331
- “Invalid context element” on page 2332
- “Missing context element” on page 2332
- “Invalid policy” on page 2332
- “Missing policy” on page 2332

If you encounter unexpected or exceptional behavior, the problem is likely related to one of these conditions. You need to examine the trace log to investigate these conditions, which requires that you configure the diagnostic trace service to generate messages about internationalization service function. To get started with logging and tracing, see Tracing and logging configuration.

The trace strings for the internationalization service follow; use both:

```
com.ibm.ws.i18n.context.*=all=enabled:com.ibm.websphere.i18n.context.*=all=enabled
```

The service is disabled

The internationalization service is not initialized when the startup setting is cleared. The service generates a message that indicates whether it is enabled or disabled. Applications cannot access the internationalization API when the service is disabled. If an application attempts a JNDI lookup to obtain the UserInternationalization reference, the lookup fails with a NamingException exception, indicating that the reference cannot be found. In addition, the service does not scope (propagate) internationalization context on incoming (outgoing) business method calls.

The service is not started

The internationalization service is operational whenever it is in the STARTED state. For example, if an application attempts to access internationalization context and the service is not started, the API issues an IllegalStateException exception. In addition, the service does not provide runtime support for servlets and enterprise beans.

As an application server progresses through its life cycle, it initializes, starts, stops, and terminates (destroys) the internationalization service. If an anomaly occurs during initialization, the service does not start. After the service is started, its state can change to BLOCKED in the event that a serious error occurs. The service generates a message for every state change.

If a trace message indicates that the service is not STARTED, examine previous messages to determine the problem. For instance, the internationalization service does not start if the activity service is unavailable and a message is displayed to that effect during initialization of the internationalization service.

During startup, the following messages indicate potential configuration or runtime problems:

No ORB support
: The service cannot obtain an instance of the object request broker (ORB). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No TCM support
: The service cannot obtain an instance of its thread context manager (TCM). This condition is a fatal error. Examine the SystemErr.log and SystemOut.log files for information.

No IIOP (activity service) support
: The service cannot register with the activity service. This condition is a fatal error. The internationalization service cannot propagate or receive context on Internet Inter-ORB Protocol (IIOP) requests without activity service support. Examine the SystemErr.log and SystemOut.log files for information.

No AsynchBeans support
: The service cannot register into the asynchronous beans environment. This warning indicates that the asynchronous beans environment cannot support internationalization context.

No EJB container support
: The service cannot register with the Enterprise JavaBeans (EJB) container. This warning indicates that the internationalization service cannot support enterprise beans. Without EJB container support, internationalization contexts do not scope properly to EJB business methods. Review the trace log for any EJB container-related error conditions.

No Web container support
: The service cannot register with the Web container. This warning indicates that the internationalization service cannot support servlets and JavaServer Page (JSP) files. Without Web container support, internationalization contexts do not scope properly to servlet service methods. Review the trace log for any Web container-related error conditions.

No Metadata support
: The service cannot register with the metadata service. This warning indicates that the internationalization service cannot process the internationalization policies within application deployment descriptors. Without metadata support, the service associates the default internationalization context management policy, [CMI, RunAsCaller], to every servlet lifecycle method and enterprise bean business method invocation. Review the trace log for any metadata service-related error conditions.

No JNDI (Naming service) support
: The service cannot bind the UserInternationalization object into the namespace. This condition is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements. Review the trace log for any Naming (JNDI) service-related error conditions.

No API support
: The service cannot obtain an instance of an internationalization context API object. This condition

is a fatal error. Application components are unable to access internationalization context API references, and are therefore unable to access internationalization context elements.

Invalid context element

The service detected an invalid internationalization context element. For example, the internationalization service does not support TimeZone instances of a type other than java.util.SimpleTimeZone. If the service encounters an unusable element, it logs a message and substitutes the corresponding default element of the JVM.

Missing context element

The service detected a missing internationalization context element. Incoming requests (for example, from application servers that do not support the internationalization service) lack internationalization context. When the service attempts to access a caller internationalization context element (which does not exist in this case), the service logs a message and substitutes the corresponding default element of the Java virtual machine (JVM).

Whenever possible, enable the internationalization service within all clients and hosting application servers that comprise an internationalized enterprise application. Read more information about Administering the internationalization service in the *Administering applications and their environment* PDF book.

Invalid policy

The internationalization service detected a malformed internationalization policy in the application deployment descriptor. The service replaces the malformed attribute with the appropriate default. For instance, if the internationalization type for an entity bean is set to `Application` during the run of a servlet or EJB business method call, the service logs the inconsistency and enforces the `Container` setting instead.

Also, AMI application components do have an implicit container internationalization attribute. By default they run as server. The service silently enforces the implicit policy, `[AMI, RunAsServer]`, and logs messages to this effect.

Invalid container internationalization attributes are likely to occur when specifying the Locales and Time zone ID fields. When encountering invalid locales and time zone IDs within attributes, the service replaces each value with the corresponding default element of the JVM. Be sure to follow the guidelines provided in the *Developing and deploying applications* PDF book.

Missing policy

The service detected a missing internationalization policy. The service replaces the missing policy with the appropriate default. For instance, if the internationalization type is missing for a servlet or enterprise bean, the service sets the attribute to Container.

Container internationalization attributes are not mandatory for CMI application components. In the event that a CMI servlet or EJB business method lacks a container internationalization attribute, the service silently enforces the implicit policy `[CMI, RunAsCaller]`.

When an application lacks internationalization policies in its deployment descriptor, or metadata support is unavailable, the service logs a message and applies the policy `[CMI, RunAsCaller]` on every servlet service method and EJB business method invocation.

Read the information in the *Developing and deploying applications* PDF book:
- Assembling internationalized applications
- Container internationalization attributes

- Internationalization type

Object pools

Using object pools

An object pool helps an application avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused.

About this task

Object pools are not meant to be used for pooling JDBC connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To use an object pool, the product administrator must define an *object pool manager* using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Note: The Object pool manager service is only supported from within the EJB container or Web container. Looking up and using a configured object pool manager from a Java 2 Platform Enterprise Edition (J2EE) application client container is not supported.

1. Start the administrative console.
2. Click **Resources > Object pool managers**.
3. Specify a **Scope** value and click **New**.
4. Specify the required properties for work manager settings.
   Scope The scope of the configured resource. This value indicates the location for the configuration file.
   Name The name of the object pool manager. This name can be up to 30 ASCII characters long.
   JNDI Name
   The Java Naming and Directory Interface (JNDI) name for the pool manager.
5. [Optional] Specify a **Description** and a **Category** for the object pool manager.

Results

After you have completed these steps, applications can find the object pool manager by doing a JNDI lookup using the specified JNDI name.

Example

The following code illustrates how an application can find an object pool manager object:

```
InitialContext ic = new InitialContext();
ObjectPoolManager opm = (ObjectPoolManager)ic.lookup("java:comp/env/pool");
```

When the application has an ObjectPoolManager, it can cache an object pool for classes of the types it wants to use. The following is an example:

```
ObjectPool arrayListPool = null;
ObjectPool vectorPool = null;
try
{
 arrayListPool = opm.getPool(ArrayList.class);
 vectorPool = opm.getPool(Vector.class);
}
catch(InstantiationException e)
{
 // problem creating pool
```

```
}
catch(IllegalAccessException e)
{
 // problem creating pool
}
```

When the application has the pools, the application can use them as in the following example:

```
ArrayList list = null;
try
{
 list = (ArrayList)arrayListPool.getObject();
 list.clear(); // just in case
 for(int i = 0; i < 10; ++i)
 {
  list.add("" + i);
 }
 // do what ever we need with the ArrayList
}
finally
{
 if(list != null) arrayListPool.returnObject(list);
}
```

This example presents the basic pattern for using object pooling. If the application does not return the object, then the only adverse effect is that the object cannot be reused.

Object pool managers

Object pool managers control the reuse of application objects and Developer Kit objects, such as Vectors and HashMaps.

Multiple object pool managers can be created in an Application Server cell. Each object pool manager has a unique cell-wide Java Naming and Directory Interface (JNDI) name. Applications can find a specific object pool manager by doing a JNDI lookup using the specific JNDI name.

The object pool manager and its associated objects implement the following interfaces:

```
public interface ObjectPoolManager
{
 ObjectPool getPool(Class aClass)
  throws InstantiationException, IllegalAccessException;
 ObjectPool createFastPool(Class aClass)
  throws InstantiationException, IllegalAccessException;

}

public interface ObjectPool
{
 Object getObject();
 void returnObject(Object o);

}
```

The getObject() method removes the object from the object pool. If a getObject() call is made and the pool is empty, then an object of the same type is created. A returnObject( ) call puts the object back into the object pool. If returnObject() is not called, then the object is no longer allocatable from the object pool. If the object is not returned to the object pool, then it can be garbage collected.

Each object pool manager can be used to pool any Java object with the following characteristics:

- The object must be a public class with a public default constructor.
- If the object implements the java.util.Collection interface, it must support the optional clear() method.

Each pooled object class must have its own object pool. In addition, an application gets an object pool for a specific object using either the ObjectPoolManager.getPool() method or the ObjectPoolManager.createFastPool() method. The difference between these methods is that the getPool() method returns a pool that can be shared across multiple threads. The createFastPool() method returns a pool that can only be used by a single thread.

If in a Java virtual machine (JVM), the getPool() method is called multiple times for a single class, the same pool is returned. A new pool is returned for each call when the createFastPool() method is called. Basically, the getPool() method returns a pool that is thread-synchronized.

The pool for use by multiple threads is slightly slower than a fast pool because of the need to handle thread synchronization. However, extreme care must be taken when using a fast pool.

Consider the following interface:

```
public interface PoolableObject
{
 void init();
 void returned();
}
```

If the objects placed in the pool implement this interface and the ObjectPool.getObject() method is called, the object that the pool distributes has the init() method called on it. When the ObjectPool.returnObject() method is called, the PoolableObject.returned() method is called on the object before it is returned to the object pool. Using this method objects can be pre-initialized or cleaned up.

It is not always possible for an object to implement PoolableObject. For example, an application might want to pool ArrayList objects. The ArrayList object needs clearing each time the application reuses it. The application might extend the ArrayList object and have the ArrayList object implement a poolable object. For example, consider the following:

```
public class PooledArrayList extends ArrayList implements PoolableObject
{
 public PooledArrayList()
 {
 }

 public void init() {
 }

 public void returned()
 {
  clear();
 }
}
```

If the application uses this object, in place of a true ArrayList object, the ArrayList object is cleared automatically when it is returned to the pool.

Clearing an ArrayList object simply marks it as empty and the array backing the ArrayList object is not freed. Therefore, as the application reuses the ArrayList, the backing array expands until it is big enough for all of the application requirements. When this point is reached, the application stops allocating and copying new backing arrays and achieves the best performance.

It might not be possible or desirable to use the previous procedure. An alternative is to implement a custom object pool and register this pool with the object pool manager as the pool to use for classes of that type. The class is registered by the WebSphere administrator when the object pool manager is defined in the cell. Take care that these classes are packaged in Java Archive (JAR) files available on all of the nodes in the cell where they might be used.

Object pool managers collection

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers**.

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Name:

Specifies the name by which the object pool manager is known for administrative purposes.

Data type	String
Range	1 through 30 ASCII characters

JNDI name:

Specifies the Java Naming and Directory Interface (JNDI) name for the object pool manager.

Data type	String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Description:

Specifies the description of the object pool manager.

Data type	String

Category:

Specifies the category name used to classify or group this object pool manager.

Data type	String

Object pool managers settings:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name*

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

The name by which the object pool manager is known for administrative purposes.

Data type	String
Range	1 through 30 ASCII characters

JNDI Name:

The Java Naming and Directory Interface (JNDI) name for the object pool manager.

Data type	String

Description:

A description of the object pool manager.

Data type	String

Category:

A category name used to classify or to group this object pool manager.

Data type	String

Custom object pool collection:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name* **> Custom object pools**.

Use custom object pools to insert additional logic around the following mechanisms:

- Constructing an object pool (A list of properties can be set)
- Flushing the object pool
- Getting objects from the pool
- Returning objects from the pool

These features allow for actions such as, clearing the state of an object when returning it to the pool, configuring the state of an object when retrieving it from the pool, or configuring generic pools and sending instructions on how to behave using custom properties.

To use an object pool the product administrator must define an object pool manager using the administrative console. You can create multiple object pool managers in an Application Server cell.

Pool class name:

Specifies the fully qualified class name of the objects that are stored in the custom object pool.

Data type String

Pool implementation class name:

Specifies the fully qualified class name of the implementation class for the custom object pool.

Data type String

Custom object pool settings:

An object pool manages a pool of arbitrary objects and helps applications avoid creating new Java objects repeatedly. Most objects can be created once, used and then reused. An object pool supports the pooling of objects waiting to be reused. These object pools are not meant to be used for pooling Java Database Connectivity connections or Java Message Service (JMS) connections and sessions. WebSphere Application Server provides specialized mechanisms for dealing with those types of objects. These object pools are intended for pooling application-defined objects or basic Developer Kit types.

To view this administrative console page, click **Resources > Object pool managers >** *objectpoolmanager_name* **> Custom object pools >** *objectpool_name*.

Use custom object pools to insert additional logic around the following mechanisms:

- Constructing an object pool (A list of properties can be set)
- Flushing the object pool
- Getting objects from the pool
- Returning objects from the pool

These features allow for actions such as, clearing the state of an object when returning it to the pool, configuring the state of an object when retrieving it from the pool, or configuring generic pools and sending instructions on how to behave using custom properties.

To use an object pool, the product administrator must define an object pool manager using the administrative console. Multiple object pool managers can be created in an Application Server cell.

Pool Class Name:

The fully qualified class name of the objects that are stored in the object pool.

Data type String

Pool Impl Class Name:

The fully qualified class name of the CustomObjectPool implementation class for this object pool.

Data type String

Object pool service settings

Use this page to enable or disable the object pool service, which manages object pool resources used by the server.

To view this administrative console page, click **Servers > Application Servers >** *server_name* > Container services > **Object Pool Service**.

Enable service at server startup:

Specifies whether the server attempts to start the object pool service.

Default Cleared

Range

Selected
When the application server starts, it attempts to start the object pool service automatically.

Cleared
The server does not try to start the object pool service. If object pool resources are used on this server, then the system administrator must start the object pool service manually or select this property, and then restart the server.

Object pools: Resources for learning

This topic provides links to find relevant supplemental information about object pools.

Use the following links to find relevant supplemental information about object pools. The information resides on IBM and non-IBM Internet sites, whose sponsors control the technical accuracy of the information.

These links are provided for convenience. Often, the information is not specific to the IBM WebSphere Application Server product, but is useful all or in part for understanding the product. When possible, links are provided to technical papers and Redbooks that supplement the broad coverage of the release documentation with in-depth examinations of particular product areas.

Furthermore, these links provide guidance on using object pools. Since object pooling is a general topic and the WebSphere Application Server product implementation is only one way to use it, you must understand when object pooling is necessary. These articles help you make that decision.

Programming model and decisions

- Build your own ObjectPool in Java to boost application speed
- Improve the robustness and performance of your ObjectPool
- Recycle broken objects in resource pools

MBeans for object pool managers and object pools

Legacy MBean names for object pool managers and object pools are deprecated. The legacy names are based on the object pool manager name (which is not required to be unique) rather than the object pool manager JNDI name.

About this task

For object pools, the legacy name is also lacking any identifier of the version of the pooled class. Additionally, object pool Performance Monitoring Instrumentation (PMI) statistics are aggregated for object pools with the same legacy object pool MBean name.

For example, if the object pool manager and pooled class are as follows:

```
object pool manager name:               My ObjectPool
object pool manager JNDI name:          op/MyObjectPool
pooled class name:                      java.util.ArrayList
hash code of java.util.ArrayList.class: 1111eb3f (hexadecimal)
```

the legacy object pool manager MBean name will be:

```
ObjectPoolManager_My ObjectPool
```

and the legacy object pool MBean name will be:

```
ObjectPool_My ObjectPool_java.util.ArrayList
```

Instead of using the deprecated legacy MBean names, use the MBean names that are based on the JNDI name of the object pool manager.

For the example above, the JNDI name-based object pool manager MBean name is:

```
ObjectPoolManager_op/MyObjectPool
```

and the JNDI name-based object pool MBean name is:

```
ObjectPool_op/MyObjectPool_java.util.ArrayList.class@1111eb3f
```

Formats for MBean names

Type	Name format
Deprecated legacy object pool manager MBean name:	ObjectPoolManager_[object pool manager name]
JNDI name-based object pool manager MBean name:	ObjectPoolManager_[object pool manager JNDI name]
Deprecated legacy object pool MBean name:	ObjectPool_[object pool manager name]_[pooled class name]
JNDI name-based object pool MBean name:	ObjectPool_[object pool manager JNDI name]_[pooled class name].class@[hexadecimal representation of the hash code of the pooled class' java.lang.Class reference]

In all of the above formats, characters that are not valid for MBean names are replaced with the '.' character.

MBeans for object pool managers and object pools

Legacy MBean names for object pool managers and object pools are deprecated. The legacy names are based on the object pool manager name (which is not required to be unique) rather than the object pool manager JNDI name.

About this task

For object pools, the legacy name is also lacking any identifier of the version of the pooled class. Additionally, object pool Performance Monitoring Instrumentation (PMI) statistics are aggregated for object pools with the same legacy object pool MBean name.

For example, if the object pool manager and pooled class are as follows:

```
object pool manager name:               My ObjectPool
object pool manager JNDI name:          op/MyObjectPool
pooled class name:                      java.util.ArrayList
hash code of java.util.ArrayList.class: 1111eb3f (hexadecimal)
```

the legacy object pool manager MBean name will be:

```
ObjectPoolManager_My ObjectPool
```

and the legacy object pool MBean name will be:

```
ObjectPool_My ObjectPool_java.util.ArrayList
```

Instead of using the deprecated legacy MBean names, use the MBean names that are based on the JNDI name of the object pool manager.

For the example above, the JNDI name-based object pool manager MBean name is:

```
ObjectPoolManager_op/MyObjectPool
```

and the JNDI name-based object pool MBean name is:

```
ObjectPool_op/MyObjectPool_java.util.ArrayList.class@1111eb3f
```

Formats for MBean names

Type	Name format
Deprecated legacy object pool manager MBean name:	ObjectPoolManager_[object pool manager name]
JNDI name-based object pool manager MBean name:	ObjectPoolManager_[object pool manager JNDI name]
Deprecated legacy object pool MBean name:	ObjectPool_[object pool manager name]_[pooled class name]
JNDI name-based object pool MBean name:	ObjectPool_[object pool manager JNDI name]_[pooled class name].class@[hexadecimal representation of the hash code of the pooled class' java.lang.Class reference]

In all of the above formats, characters that are not valid for MBean names are replaced with the '.' character.

Scheduler

Using schedulers

Schedulers enable Java Platform, Enterprise Edition (Java EE) application tasks to run at a requested time. Schedulers also enable application developers to create their own stateless session Enterprise JavaBeans (EJB) components to receive event notifications during a task life cycle, allowing the plugging-in of custom logging utilities or workflow applications.

About this task

You can schedule the following types of tasks:
- Invoke a session bean method
- Send a Java Message Service (JMS) message to a queue or topic

Stateless session EJB components are also used to provide generic calendaring. Developers can either use the supplied calendar bean or create their own for their existing business calendars. For example, one of your business processes might involve invoicing for services. With the scheduler's use of stateless EJB components, you can schedule when periodic email distributions are to be sent to your customers who have received invoices. The scheduler service performs these tasks, repeating as necessary, according to the metadata for that task.

A scheduler is the mechanism by which the timer service for Enterprise Java Beans 2.1 runs. You can configure the EJB timer service to use many of the features that schedulers provide See the documentation for more details.

Use the following table to determine which persistent timer service is best for you:

Schedulers	EJB timers
Run stateless session EJB components and sends JMS messages	Run all EJB types except for stateful session beans
Persistent, transactional and highly available.	Persistent, transactional and highly available.
Tasks guaranteed to run only once	Timers guaranteed to run only once, if the timer EJB uses a container-managed global transaction
Run repeating tasks using any calculation rules	Run repeating tasks using a repeating interval defined in milliseconds
Uses a modified fixed-delay time calculation to determine repeating intervals (next run time based on the start-time of the previous task)	Uses a fixed-rate time calculation to determine repeating intervals (time of the next task is based on the original scheduled time).
Programmatic task monitoring capability with the use of the NotificationSink stateless session EJB component	No programmatic timer monitoring
Abort late or time-sensitive tasks from running	Abort late or time-sensitive tasks from running (achieved through manual detection within the javax.ejb.TimedObject implementation).
Manage any task lifecycle (find, suspend, resume, cancel and purge tasks programmatically and through Java Management Extensions (JMX)).	Find and cancel its timers programmatically. Administrators find and cancel timers using a command-line utility.
Store a limited amount of text with the data, like a **Name** (arbitrary data stored externally).	Store arbitrary data with a timer

This task demonstrates how to manage, develop and interoperate with schedulers and subsequent tasks.

1. Manage the scheduler service. This topic includes instructions for creating and configuring schedulers, creating and configuring a database for schedulers and administering schedulers.
2. Develop and schedule tasks. This topic includes instructions for developing various types of tasks, receiving notifications from a task, submitting tasks to a scheduler, and managing tasks.

   Note: Creating and manipulating scheduled tasks through the Scheduler API interface is only supported from within the Enterprise Java Beans (EJB) container or Web container (JavaServer Pages or servlets). Looking up and using a configured scheduler from a Java EE application client container is not supported.
3. Interoperate with schedulers. This topic explains how to manage scheduler in a clustered environment with mixed WebSphere Application Server product versions and mixed platforms.

Example: Using default scheduler calendars

The SIMPLE and CRON calendars can be used from any J2EE application. This topic describes that process.

Using default scheduler calendars involves looking-up the default UserCalendarHome Enterprise JavaBeans (EJB) home object, creating the UserCalendar bean and calling the applyDelta() method. For details on the applyDelta method as well as the syntax for the SIMPLE and CRON calendars, see the UserCalendar interface topic.

Example:

```
import java.util.Date;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import com.ibm.websphere.scheduler.UserCalendar;
import com.ibm.websphere.scheduler.UserCalendarHome;
```

```
// Create an initial context
InitialContext ctx = new InitialContext();

// Lookup and narrow the default UserCalendar home.
UserCalendarHome defaultCalHome=(UserCalendarHome)
 PortableRemoteObject.narrow(ctx.lookup(
      UserCalendarHome.DEFAULT_CALENDAR_JNDI_NAME),
    UserCalendarHome.class);

// Create the default UserCalendar instance.
UserCalendar defaultCal = defaultCalHome.create();

// Calculate a date using CRON based on the current
// date and time.  Return the next date that is
// Saturday at 2AM
Date newDate =
    defaultCal.applyDelta(new Date(),
        "CRON", "0 0 2 ? * SAT");
```

Scheduler daemon

A scheduler daemon is a background thread that searches for tasks to run in the database.

A scheduler daemon is started for each scheduler defined on each server. If `Scheduler 1` is configured on `server1`, then only one scheduler daemon runs on `server1` unless it is cloned. If `Scheduler 1` is defined at the node scope level, then the scheduler will run on each server within that node.

The poll interval determines the frequency at which the persistent store is queried. By default, this value is set to 30 seconds. When a task is found that is scheduled to run within the current poll interval, an asynchronous beans alarm is set. The task then runs as close to this time as possible using an alarm thread from the scheduler's associated work manager. Thus, the number of alarm threads configured on the work manager determines how many concurrent tasks are executed. No tasks are lost. If we reach this limit, then new tasks are simply queued to be executed when an alarm thread becomes available. The actual firing time is dictated by server load and availability of free threads in the alarm thread pool of the associated work manager.

Scheduler daemons in a cluster

When multiple schedulers are configured to use the same tables (as is the case in a clustered environment), any of the daemons can find a task and set the alarm in its Java virtual machine (JVM). The task is executed in the virtual machine where the scheduler daemon first runs, until the daemon is stopped and another daemon starts. If an application on `server1` schedules a task to run and `server2` was started before `server1`, then the task runs on `server2`.

Example: Stopping and starting scheduler daemons using Java Management Extensions API:

Use the wsadmin scripting tool to invoke a `Jacl` script and stop and start a scheduler daemon.

This example JACL script can be invoked using the wsadmin scripting tool. It will attempt to stop and start a scheduler daemon.

```
# Example JACL Script to restart a Scheduler Daemon

set schedJNDIName sched/MyScheduler

# Find the WASScheduler MBean
regsub -all {/} $schedJNDIName "." schedJNDIName
set mbeanName Scheduler_$schedJNDIName
puts "Looking up Scheduler MBean $mbeanName"
set sched [$AdminControl queryNames WebSphere:*,type=WASScheduler,name=$mbeanName]

# Invoke the stopDaemon operation.
puts "Stopping the daemon..."
```

```
$AdminControl invoke $sched stopDaemon
puts "The daemon has stopped."

# Invoke the startDaemon operation.
puts "Starting the daemon..."
$AdminControl invoke $sched startDaemon 0
puts "The daemon has started."
```

Example: Dynamically changing scheduler daemon poll intervals using Java Management Extensions API:

Use the wsadmin scripting tool to invoke a `Jacl` script and dynamically change scheduler daemon poll intervals.

To dynamically change scheduler daemon poll intervals, use the wsadmin scripting tool to invoke this example `JACL` script. Invoking this example sets the poll interval of the scheduler daemon to 60 seconds.

```
# Example JACL Script to set the Scheduler daemon's poll interval

set schedJNDIName sched/MyScheduler

# Find the WASScheduler MBean
regsub -all {/} $schedJNDIName "." schedJNDIName
set mbeanName Scheduler_$schedJNDIName
puts "Looking-up Scheduler MBean $mbeanName"
set sched [$AdminControl queryNames WebSphere:*,type=WASScheduler,name=$mbeanName]

# Set the poll interval to 60 seconds (60000 ms)
$AdminControl setAttribute $sched pollInterval 60000
puts "Poll interval set."
```

Interoperating with schedulers

Schedulers support forward compatibility. Tasks created in previous versions of WebSphere Application Server Enterprise Edition 5.0 or WebSphere Business Integration Server Foundation 5.1 continue to run in WebSphere Application Server, Version 6.x schedulers. Tasks that you create using Version 6.x are not compatible with product schedulers from Version 5.x. Version 5.x schedulers do not run any Version 6.x tasks.

Schedulers and versions

All schedulers that are configured to use the same database and tables are considered a clustered scheduler. To guarantee that your tasks run correctly, all servers in a scheduler cluster must be at the same version. If the servers are at different versions, tasks created with a Version 6.x scheduler might not run. If a mixed-Version environment is required for a short period of time, then all scheduler poll daemons should be stopped on all Version 5.x servers to allow a Version 6.x server to run all tasks. This action allows the Version 6.x schedulers to obtain leases and run tasks that have been created with a Version 6.x scheduler.

Running tasks created with schedulers prior to Version 5.0.2 is not supported. See the topic, ″Interoperating with the Scheduler service,″ in the WebSphere Application Server Enterprise Edition Version 5.0.2 information center for details on how to migrate these tasks to a more recent version. See the Information Center Library to access the Version 5.0.2 information center.

Scheduler calendars

The scheduler provides stateless session bean interfaces which allow creating common calendars which can be used by the scheduler and any Java Platform, Enterprise Edition (Java EE) application.

The `SchedulerCalendars.ear` application is available and provides a default UserCalendar Enterprise Java Beans (EJB) implementation which allows using the SIMPLE and CRON calendars. Although this application is not required when using the scheduler, it is available to use from any Java EE application.

For details on how the SIMPLE and CRON calendars behave, see the API documentation for the com.ibm.websphere.scheduler.UserCalendar interface.

Specifying a UserCalendar with the scheduler

A UserCalendar is specified using the setUserCalendar() method of the TaskInfo interface of the scheduler. This interface allows you to select the Java Naming and Directory Interface (JNDI) name of the home interface of a UserCalendar bean. Because some UserCalendar bean implementations might handle multiple types of calendars, the interface also allows you to optionally select which type of calendar to use. A list of valid calendar types can be retrieved by invoking the getCalendarNames() method of the UserCalendar interface.

If the setUserCalendar() method is not invoked, or if a value of null or empty-string is specified for the home JNDI name parameter, then the default UserCalendar is used internally by the scheduler. When the default UserCalendar is accessed internally, it is not necessary that the `SchedulerCalendars.ear` system application be installed.

You might want to use the default UserCalendar directly in your other Java EE applications, apart from the scheduler. In this case, you may use the UserCalendarHome.DEFAULT_CALENDAR_JNDI_NAME value to look up the default UserCalendar from your applications. You may also supply this value to the setUserCalendar() method of the TaskInfo interface. You will need to ensure the `SchedulerCalendars.ear` system application was either automatically installed or that you have installed it manually.

Scheduler service settings

Use this page to enable or disable the scheduler service. The scheduler service manages scheduler resources used by the server. The administrative console page used to configure the scheduler service is not available for version 6 (and above) servers. It is only available for version 5.*x* servers.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Scheduler Service**.

Startup:

Specifies whether the server attempts to start the scheduler service.

Default
Selected

Range

Selected
When the application server starts, it attempts to start the scheduler service automatically.

Cleared
The server does not try to start the scheduler service. If scheduler resources are to be used on this server, the system administrator must start the scheduler service manually or select this property, then restart the server.

Installing default scheduler calendars

The default scheduler SIMPLE and CRON calendars are available in the `SchedulerCalendars.ear` system application and are automatically installed on standalone server profiles. System applications cannot be installed and uninstalled like traditional Java Platform, Enterprise Edition (Java EE) applications.

About this task

The following steps are required to map the `SchedulerCalendars.ear` system application on a server or cluster in a network deployment environment.

1. Start the wsadmin tool and connect to the deployment manager.
2. Install the system application.
   - To install on a non-clustered server:
     - Using Jacl:

       ```
       $AdminApp install
       "\${WAS_INSTALL_ROOT}/systemApps/SchedulerCalendars.
       ear" {-systemApp -appname SchedulerCalendars -cell
        mycell -node mynode -server myserver}
       ```

     - Using Jython list:

       ```
       AdminApp.install('${WAS_INSTALL_ROOT}/systemApps/
       SchedulerCalendars.ear', ['-systemApp', '–appName',
        'SchedulerCalendars','-cell', 'mycell', '-node',
        'mynode', '-server', 'myserver'])
       ```

     - Using Jython string:

       ```
       AdminApp.install('${WAS_INSTALL_ROOT}/systemApps/
       SchedulerCalendars.ear', '[-systemApp –appName
        SchedulerCalendars –cell mycell –node mynode
        -server myserver]')
       ```

       Where:

Value	Option
mycell	the value of the cell option
mynode	the value of the node option
myserver	the value of the server option

   - To install on a cluster:
     - Using Jacl:

       ```
       $AdminApp install
       "\${WAS_INSTALL_ROOT}/systemApps/SchedulerCalendars.
       ear" {-systemApp -appname SchedulerCalendars -cell
        mycell -cluster mycluster}
       ```

     - Using Jython list:

       ```
       AdminApp.install('${WAS_INSTALL_ROOT}/systemApps/
       SchedulerCalendars.ear', ['-systemApp', '–appName',
        'SchedulerCalendars','-cell', 'mycell', '-cluster',
        'mycluster'])
       ```

     - Using Jython string:

       ```
       AdminApp.install('${WAS_INSTALL_ROOT}/systemApps/
       SchedulerCalendars.ear', '[-systemApp –appName
        SchedulerCalendars –cell mycell -cluster mycluster]')
       ```

       Where:

Value	Option
mycell	the value of the cell option
mycluster	the value of the cluster option

3. Save the configuration changes.

Uninstalling default scheduler calendars

The default scheduler SIMPLE and CRON calendars are available in the `SchedulerCalendars.ear` system application and are automatically installed on standalone server profiles. System applications cannot be installed and uninstalled like traditional Java Platform, Enterprise Edition (Java EE) applications.

About this task

Remove the SchedulerCalendars system application on federated node, as follows:

1. Open a command window on the federated node.
2. Run the following command:

   On Unix platforms:

   ```
   $Install_Root/bin/wsadmin.sh -conntype none -profile $Profile_Name
   ```

   On Windows platforms:

   ```
   $Install_Root/bin/wsadmin -conntype none -profile $Profile_Name
   ```

   where:

   - *Install_Root* is the directory where WebSphere Application Server is installed.
   - *Profile_Name* is the name of the profile where the target server is located.
3. At the **wsadmin>** prompt, enter the following command for each server that exists on the node where you want to have the SchedulerCalendars application available:

   ```
   wsadmin> $AdminApp uninstall SchedulerCalendars "-cell $MyCell -node $MyNode -server $MyServer"
   ```

   where:

   - *$Install_Root* is the directory where WebSphere Application Server is installed.
   - *$MyCell*, *$MyNode*, and *$MyServer* are the values with the name of the cell, node, and server.

   Note: Each of these values are case-sensitive.
4. Repeat step three for each server in the current profile for which you will uninstall the SchedulerCalendars application.
5. When uninstallation is complete for the system application on all appropriate servers, enter the following commands:

   ```
   wsadmin> $AdminConfig save
   wsadmin> exit
   ```
6. Using the Administrative Console or scripting, start or restart the servers to unload the SchedulerCalendars application.

Results

The SchedulerCalendars application should now be removed.

Managing schedulers

Schedulers are configured using the administrative console, configuration service or scripting and are available to all servers on which a scheduler is visible.

About this task

You can create multiple schedulers within a single server, cluster, node or cell. Each configured scheduler is an independent task scheduling engine that has a unique Java Naming and Directory Interface (JNDI) name, persistent storage device and daemon.

1. Configure schedulers.
2. Create the database for schedulers.

Configuring schedulers

Before your application can make use of the scheduler service, you must configure a scheduler using the administrative console, configuration service or scripting. Conceptually, a scheduler is similar to a data source in that you must specify various configuration attributes, including a JNDI name where the instance

is bound. Once defined, an application using the Scheduler API or WASScheduler MBean can look up the scheduler object and call various methods to manage tasks.

About this task

The scheduler service is always enabled. In previous versions of the product, the scheduler service could be disabled using the administrative console or configuration service. Scheduler service configuration objects are present in the configuration service, but the enabled attribute is ignored.

To achieve high availability, you can configure a duplicate scheduler on each server in a cluster, or create a scheduler at the cluster scope. For example, each server that contains a scheduler with the JNDI name `sched/MyScheduler`, with the same database configuration parameters (data source and table prefix) behaves as a single clustered scheduler. Each server in the scheduler cluster has a running scheduler instance, which increases the number of poll daemons and allows automatic failover. For more information on creating clusters for high availability, see the article, ″WebSphere Enterprise Scheduler planning and administration guide.″

Typically, create schedulers at the server or cluster scope. Scheduler poll daemons run in each server within the configured scope, which means that if you create a scheduler at the node or cell scope, the scheduler poll daemon can attempt to run tasks on any of the servers in the node or cell. If applications are not mapped uniformly over each server in that scope, the scheduler might not run tasks correctly. Because applications are mapped to servers and clusters, there is less chance for error and less competition between daemons to run tasks.

Depending on your preferred method of configuration, select one of the following steps to configure schedulers.

1. Configuring schedulers using the administrative console.
2. Configuring schedulers using Java Management Extensions API (JMX).

Results

A scheduler is configured and ready to use.

Configuring scheduler default transaction isolation:

The scheduler uses read-committed transaction isolation, by default, when reading tasks using the get or find APIs on the com.ibm.websphere.scheduler.Scheduler interface and WASScheduler MBean. The default behavior for a scheduler can be changed to read-uncommitted, which allows the get and find methods to return the current or next state of the task in the database. This topic describes how to change the default behavior for the get and find methods.

About this task

See the scheduler API documentation to view the com.ibm.websphere.scheduler.TaskInfo.setTaskExecutionOptions() method, which details how to return the next state of the task or the current state of the task.

Note: If the scheduler database does not support uncommitted reads, such as Oracle, this parameter has no effect.

To change the default behavior for the get and find methods, complete the following steps:

1. From the administrative console, click **Resources** > **Schedulers** > *scheduler_name*.
2. Click **Custom Properties**.
3. Click **New**.

4. Add the following properties:

Name	defaultReadTransactionIso
Type	java.lang.Integer
Value	1 (for read-uncommitted transaction isolation) 2 (for read-committed transaction isolation)

5. Click **Apply** or
6. Click **OK**.
7. Save the changes and verify that you initiate a file synchronization before restarting the servers.
8. Restart the application server for the changes to take effect.

Configuring schedulers using the administrative console:

Schedulers can be created or configured using the administrative console.

1. Start the administrative console.
2. Select **Resources** > **Schedulers**.
3. Click **New**.
4. Specify configuration settings. Fields marked with an asterisk (*) are required. The settings are described in detail in the topic ″Scheduler settings.″
5. Click **OK** or **Apply** to save the changes.
6. Save the changes to the configuration repository.

Results

A scheduler is now configured and ready to use for newly installed applications. If the scheduler JNDI name is not yet visible to your application, restarting the application or restarting application server will allow the scheduler to be seen.

When schedulers are created for the first time, the poll daemon will not automatically start and must be started manually and will only start automatically the next time the server is started. To start the poll daemon manually, see the scheduler daemons topic.

Note: Changes to existing scheduler configurations will not take affect until after the application server is restarted.

Schedulers collection:

Use this page to manage scheduler configurations. Schedulers are persistent and transactional timer services that can run business logic. Each scheduler runs tasks independently and has a programming interface accessible from Java Platform, Enterprise Edition (Java EE) applications using the Java Naming and Directory Interface (JNDI). You can also manage schedulers using a Java Management Extensions (JMX) MBean. See the scheduler documentation in the Information Center for details on how to configure and use schedulers.

To view this administrative console page, click **Resources > Schedulers**.

Name:

Specifies the name of the data source where persistent tasks are stored.

Data type String

JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of Java EE context applied to the task.

The JNDI name specifies where this scheduler instance is bound in the name space. Clients can look this name up directly, although the use of resource references is recommended.

Data type	String

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Data source JNDI name:

Specifies the alias for the user name and password that are used to access the data source.

Any data source available in the name space can be used with a scheduler. Multiple schedulers can share a single data source while using different tables by specifying a table prefix.

Data type	String

Table prefix:

Specifies the string prefix to affix to the scheduler tables.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Data type	String

Poll interval:

Specifies the interval, in seconds, that a scheduler polls the database. The default value is appropriate for most applications.

Each poll operation can be consuming. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type	Integer
Units	Seconds
Default	30
Range	Any positive long integer

Work manager JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of Java EE context applied to the task.

The work manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler uses the **Number of alarm threads** specified in the work

manager, which affects the number of tasks that can run concurrently. Use the work manager **Service Names** property to limit the amount of context information that is propagated to the task when it runs.

When a task runs, the task is run in the work manager associated with the scheduler instance. You can control the number of actively running tasks at a given time by configuring schedulers with a specific work manager. The number of tasks that can run concurrently is governed by the **Number of alarm threads** parameter on the work manager.

Note: When you configure a scheduler resource on a Version 5.x node or server, the scheduler must reference a work manager in the same scope. For example, a scheduler instance configured at the server1 scope must use a work manager also configured at the server1 scope.

Verify tables:

Specifies to validate that scheduler data sources, table prefixes, security authentication information and tables are configured correctly.

You can use this verification method in production and development environments without altering database properties.

Create tables:

Specifies to create the necessary tables and indices required for a scheduler to operate.

This method of creating scheduler tables is designed for simple topologies and development environments. Use the supplied scheduler data definition language files for advanced or production environments and for databases that do not support this feature. .

Drop tables:

Specifies the removal of tables and indices required for schedulers to operate.

This method of removing scheduler tables and indices is recommended for development environments and does not delete previously scheduled tasks.

Schedulers settings:

Use this page to modify scheduler settings.

To view this administrative console page, click **Resources > Schedulers >** *scheduler_name*.

Scope:

Specifies the scope of the configured resource. This value indicates the location for the configuration file.

Name:

Specifies the name by which this scheduler is known for administrative purposes.

Data type String

JNDI name:

Specifies the name of the data source where persistent tasks are stored.

The Java™ Naming and Directory Interface (JNDI) name specifies where this scheduler instance is bound in the namespace. Clients can look this name up directly, although the use of resource references is recommended.

Data type String

Description:

Specifies the description of this scheduler for administrative purposes.

Data type String

Category:

Specifies a string that can be used to classify or group this scheduler.

Data type String

Data source JNDI name:

Specifies the name of the data source where persistent tasks are stored.

Any data source available in the name space can be used with a scheduler. Multiple schedulers can share a single data source while using different tables by specifying a table prefix.

Data type String

Data source alias:

Specifies the alias for the user name and password that are used to access the data source.

Data type String

Table prefix:

Specifies the string prefix to affix to the scheduler tables. Multiple independent schedulers can share the same database if each scheduler specifies a different prefix string.

Multiple independent schedulers can share the same database if each instance specifies a different prefix string.

Note: Use a table prefix with all capital characters. If lowercase characters are used for the table prefix, they are automatically capitalized at run time.

Data type String

Poll interval:

Specifies the interval, in seconds, that a scheduler polls the database. The default value is appropriate for most applications.

Each poll operation can be consuming. If the interval is extremely small and there are many scheduled tasks, polling can consume a large portion of system resources.

Data type	Integer
Units	Seconds
Default	30
Range	Any positive long integer

Work manager JNDI name:

Specifies the JNDI name of the work manager, which is used to manage the number of tasks that can run concurrently with the scheduler. The work manager also can limit the amount of Java Platform, Enterprise Edition (Java EE) context applied to the task.

The work manager is a server object that serves as a logical thread pool for the scheduler. Each repeating task that is created using this scheduler uses the **Number of alarm threads** specified in the work manager, which affects the number of tasks that can run concurrently. Use the work manager **Service Names** property to limit the amount of context information that is propagated to the task when it runs.

When a task runs, the task is run in the work manager associated with the scheduler instance. You can control the number of actively running tasks at a given time by configuring schedulers with a specific work manager. The number of tasks that can run concurrently is governed by the **Number of alarm threads** parameter on the work manager.

Note: When you configure a scheduler resource on a Version 5.x node or server, the scheduler must reference a work manager in the same scope. For example, a scheduler instance configured at the server1 scope must use a work manager also configured at the server1 scope.

Use administration roles:

Specifies that when this option and administrative security are both enabled, the user administration roles are enforced when the scheduler JMX commands or APIs are used to create and modify tasks. If this option is not enabled, all the users can create and modify tasks.

Schedulers require several user roles to plan for, develop, administer and operate the scheduler service: administrator, developer and operator.

Data type	check box
Default	unchecked
Range	• Operator, Administrator --Calls any of the scheduler MBean or API methods and runs any of the scheduler administrative console functions. • Monitor, Configurator --Calls the scheduler MBean or API methods, but cannot create tasks or modify the state of any tasks. Only read-only methods and properties are accepted.

Configuring schedulers using Java Management Extensions:

Schedulers can be created or configured using the Java Management Extensions (JMX) API using one of several scripting languages or Java.

About this task

To run with Java, the following JAR file needs to be present in the program class path: `com.ibm.ws.admin.client_6.1.0.jar`.

Complete these steps when using Java programs that utilize JMX.

1. Look up the host and get an administration client handle.
2. Get a configuration service handle.
3. Update the `resource-pme.xml` file using the configuration service, as needed.
   a. Find the SchedulerProvider for a given scope.
   b. Create a SchedulerConfiguration and specify all required parameters identifying the SchedulerProvider as the parent object.
4. Reload the `resource-pme.xml` file to bind the newly created scheduler into the JNDI namespace. Perform this step if you want to use the newly created scheduler immediately, without restarting the application server.
   a. Locate the DataSourceConfigHelper MBean using the name.
   b. Invoke the reload() operation.

Results

A scheduler is now configured and ready to use for newly installed applications. If the scheduler JNDI name is not yet visible to your application, reinstalling the application or restarting the application server will allow the scheduler to be seen.

When schedulers are created for the first time, the poll daemon does not automatically start, and you must start it manually. When you restart the server, the poll daemon starts automatically. To start the poll daemon manually, see scheduler daemons.

Note: Changes to existing scheduler configurations will not take affect until after the application server is restarted.

Example: Using scripting to create and configure schedulers:

Use the wsadmin scripting tool to invoke a `Jacl` script and create a SchedulerConfiguration resource.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which creates a SchedulerConfiguration resource using the DefaultWorkManager at the server scope.

```
# Example JACL Script to create a SchedulerConfiguration
# at the server scope

# Change the cell, node and server to match your environment
set cellName   MyCell
set nodeName   MyNode
set serverName server1

# We can just grab the first provider, since there is only one at the
# server scope level.

set schedProv [$AdminConfig getid /Cell:$cellName/Node:$nodeName/Server:$serverName/
  SchedulerProvider:SchedulerProvider]
if {$schedProv == ""} {
    puts "Unable to find SchedulerProvider for server: $serverName.  Aborting."
    exit
}
puts "Found a SchedulerProvider"

# Create a WorkManager for our scheduler at the server scope.
# We could use any of the other scopes as long as it is at the same
# or higher than the Scheduler's scope.

set wrkMgrProv [$AdminConfig getid /Cell:$cellName/Node:$nodeName/Server:$serverName/
  WorkManagerProvider:WorkManagerProvider/]
if {$wrkMgrProv == ""} {
    puts "Unable to find the WorkManagerProvider for server: $serverName.  Aborting."
    exit
```

```
}
puts "Found a WorkManagerProvider"

set wmName             "MyScheduler WorkManager"
set wmJNDIName         "wm/MySchedWorkManager"
set wmIsGrowable       false
set wmMaxThreads       1
set wmMinThreads       0
set wmNumAlarmThreads  10
set wmServiceNames     "com.ibm.ws.i18n;security;UserWorkArea;zos.wlm"
set wmThreadPriority   5

# Setup our DefaultWorkManager attributes
set createAttrs [subst { \
    {isGrowable $wmIsGrowable} \
    {jndiName $wmJNDIName} \
    {maxThreads $wmMaxThreads} \
    {minThreads $wmMinThreads} \
    {name "$wmName"} \
    {numAlarmThreads $wmNumAlarmThreads} \
    {serviceNames "$wmServiceNames"} \
    {threadPriority $wmThreadPriority} }]

puts "Creating a WorkManager"
$AdminConfig create WorkManagerInfo $wrkMgrProv $createAttrs
puts "WorkManager Created"

# Setup our SchedulerConfiguration attributes
set schedulerName          MyScheduler
set schedulerJNDIName      sched/MyScheduler
set datasourceJNDIName     jdbc/MySchedulerDatasource
set datasourceAlias        MySchedulerAlias
set pollInterval           30
set tablePrefix            MSCD
set useAdminRoles          true

set createAttrs [subst { \
  {name $schedulerName} \
  {datasourceJNDIName $datasourceJNDIName} \
  {datasourceAlias $datasourceAlias} \
  {jndiName $schedulerJNDIName} \
  {pollInterval $pollInterval} \
  {tablePrefix $tablePrefix} \
  {useAdminRoles true} \
  {workManagerInfoJNDIName $wmJNDIName}}]

puts "Creating a Scheduler"
$AdminConfig create SchedulerConfiguration $schedProv $createAttrs
puts "Scheduler created"

# Save the configuration
$AdminConfig save
```

Creating a scheduler resource reference:

When you define schedulers in the server configuration, the object instance is bound into the global name space under the configured Java Naming Directory Interface (JNDI) name. You can use a resource reference to avoid manually coding this JNDI name into your application. Using a resource reference allows administrators to map applications to the appropriate schedulers.

About this task

You can alternatively create a scheduler resource reference by editing the XML directly. A Scheduler resource reference is a Java Platform, Enterprise Edition (Java EE) compliant resource that uses the class

com.ibm.websphere.scheduler.Scheduler as the object type. For information regarding the XML file format, see the Java EE Specification.

1. Start an assembly tool, such as Rational Application Developer.
2. Open the Java EE perspective.
3. Open your Enterprise JavaBeans (EJB) or Web module with the Deployment Descriptor Editor.
4. Click the **Reference** tab at the bottom of the window.
5. Click **Add**.
6. Select the **Resource reference** option.
7. Click **Next**.
8. Complete the Reference fields as shown in the following properties:
   Name The reference name, for example, *sched/MyScheduler*. According to this example, the name you choose has a local reference name of **java:comp/env/*sched/MyScheduler***.
   Type Select **com.ibm.websphere.scheduler.Scheduler**, and click **OK**.
   Authentication
   Select container.
   Description
   Any relevant description.
9. Click finish.
10. (**Optional**) Enter a global JNDI name of a configured scheduler in the JNDI name field in the **Bindings** section of the **Reference** window. You can specify or override this value when you install the application.
11. Save your changes to the deployment descriptor.

Results

A scheduler resource reference is now available to use within your application

Creating the database for schedulers

Each scheduler requires a database in which to store its persistent information. Schedulers use this database for storing tasks and then running them. The choice of database and location should be determined by the application developer and server administrator.

Before you begin

Scheduler performance is ultimately limited by database performance. If you need more tasks per second, you can run the scheduler daemons on larger systems, use clusters for the session beans used by the tasks or partition the tasks by using multiple schedulers. Eventually, however, the scheduler database becomes saturated, and a larger or better-tuned database system is needed. For detailed information on scheduler topologies see the technical paper, ″WebSphere Enterprise Scheduler planning and administration guide″.

Multiple schedulers can share a database when you specify unique table prefix values in each scheduler configuration. This sharing can lower the cost of administering scheduler databases.

About this task

Complete the following steps to create scheduler databases.

1. Create a database. To create the database for a scheduler or to determine if an existing database is adequate for a scheduler, review the topic, ″Create scheduler databases″.
2. Create the scheduler tables. There are three methods for creating the tables for a scheduler:
   a. Create tables for schedulers using the administrative console. Use the administrative console to add, delete and verify database tables through your Web browser. This method is ideal for developers and simple scheduler topologies.

b. Create tables for schedulers using JMX or scripting.

   Use JMX to add, delete and verify database tables programmatically with Java or scripting. This method is ideal for automating scheduler configurations for simple scheduler topologies.

c. Create tables for schedulers using DDL files. Manually edit the DDL files through your favorite text editor, and verify that mapping between the table names and the scheduler resources and data sources is correct.

Creating scheduler databases:

The scheduler uses the scheduler database for storing and running tasks. To create a scheduler database, your database system must be installed and available.

Before you begin

The performance of schedulers is ultimately limited by the performance of the database. If you need more tasks per second, you can run the scheduler daemons on larger systems or you can use clusters for the session beans used by the tasks. Eventually, however, the task database becomes saturated and you then need a larger or better-tuned database system.

Multiple applications can share a scheduler database. This sharing can lower the cost of administering scheduler databases.

About this task

The scheduler requires a database, a JDBC provider, and a data source.

1. Create the database according to the description for your database system:
   - Creating Apache Derby databases for schedulers.
   - Creating a DB2 database for schedulers.
   - Creating a DB2 for iSeries database for schedulers.
   - Creating an Informix database for schedulers.
   - Creating a Microsoft SQL Server database for schedulers.
   - Creating an Oracle database for schedulers.
   - Creating a Sybase database for schedulers.
2. If the database is not on the same machine as your IBM WebSphere Application Server, verify that you can access the database from your application server machine.
3. Configure your JDBC provider and data source. For details, see the topic Creating and configuring a JDBC provider and data source. The JDBC driver can be either one-phase or two-phase commit depending on whether other transactions take place using other data sources, for example, while using the scheduler. The data source can represent multiple versions of the product.

Results

The database is created and ready for you to create scheduler tables.

Creating Apache Derby databases for schedulers:

This topic describes how to create Apache Derby databases for schedulers using data definition language (DDL) or structured query language (SQL) files.

About this task

To create Apache Derby databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in Java code, which results in code page conversion problems when a client uses an incompatible code page.
4. Use the ij utility supplied with the Apache Derby system to create the database. To use ij to create a database called scheddb which is located in `/opt`, for example, you would do the following:

   ```
   ij.sh
   ij>connect '/opt/scheddb;create=true';
   ij>quit;
   ```

   The embedded version of Apache Derby supports only one local connection. If the Application Server product is running and accessing a Apache Derby database, then any attempts to open a second connection to the database from the command line are rejected.

   Note: Add a semi-colon (;) at the end of each command. Otherwise, ij does not execute the command.
5. Exit the ij utility by issuing the quit command: `quit;`

Results

The Apache Derby database for the scheduler service exists.

Creating DB2 databases for schedulers:

This topic describes how to create DB2 databases for scheduler, using data definition language (DDL) or structured query language (SQL) files.

About this task

To create DB2 databases for scheduler, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a DB2 command-line window.
2. Make sure that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, it cannot store all the characters that can be handled in Java code, which might result in code page conversion problems, when a client uses an incompatible code page.

   To avoid deadlocks, be sure that the DB2 isolation level is set to ″read stability″. If necessary, enter the command

   ```
   db2set DB2_RR_TO_RS=YES
   ```

   then restart the DB2 instance to activate the change.
4. In the DB2 command line processor, enter this command to create the database with an example name, `scheddb`:

   ```
   db2 CREATE DATABASE scheddb USING CODESET UTF-8 TERRITORY en-us
   ```

   A DB2 database named `scheddb` has been created.

Results

The DB2 database for the scheduler exists.

Creating DB2 for iSeries databases for schedulers:

This topic describes how to create DB2 for iSeries databases for scheduler, using data definition language (DDL) or structured query language (SQL) files.

About this task

To create DB2 for iSeries databases for scheduler, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Run the following command to start an interactive SQL session:
   ```
   STRSQL
   ```
2. In interactive SQL, enter this command to create the collection with an example name, `scheddb`:
   ```
   CREATE COLLECTION scheddb
   ```
3. Exit the interactive SQL session.
4. Change the owner for the new collection to QEJBSVR by executing the following command:
   ```
   CHGOBJOWN OBJ(scheddb) OBJTYPE(*LIB) NEWOWN(QEJBSVR)
   ```

   where `scheddb` is the name of the collection you created in the previous step.

Results

The DB2 for iSeries database for the scheduler exists.

Creating Informix databases for schedulers:

This topic describes how to create Informix databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

About this task

To create Informix databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.
4. If you want to create a new database named `scheddb`, for example, enter the command:
   ```
   dbaccess CREATE DATABASE scheddb with log
   ```

Results

The Informix database for scheduler exists.

Creating Microsoft SQL Server databases for schedulers:

This topic describes how to create Microsoft SQL Server databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

About this task

To create Microsoft SQL Server databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Make sure that you are using a user ID that has administrator rights for the database system.

2. In the **Enterprise Manager**, expand a server group, then expand a server. A Microsoft SQL Server database named `scheddb` is created.
3. Right-click **Databases**, then click **New Database**.
4. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible codepage.
5. Type the name `scheddb`.
6. Modify any default values, and save your changes. The Microsoft SQL Server database, scheddb, is created.

Results

The Microsoft SQL Server database for scheduler exists.

Creating Oracle databases for schedulers:

This topic describes how to create Oracle databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

About this task

To create Oracle databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.
4. Use the Database Configuration Assistant to create the database, `scheddb`, for example. Verify that you select the **JServer** option for the database. Use a Unicode code page when creating the database. The text data you pass to the APIs must be compatible with the selected code page.

Results

The Oracle database for scheduler exists.

Creating Sybase databases for schedulers:

This topic describes how to create Sybase databases for schedulers, using data definition language (DDL) or structured query language (SQL) files.

About this task

This topic describes how to create Sybase databases for schedulers, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Make sure that you have the DTM option for Sybase ASE installed.
4. Verify that the database supports Unicode (UTF-8). Otherwise, the database does not store all characters that can be handled in the Java code, which results in code page conversion problems, when a client uses an incompatible code page.
5. Use the Sybase isql utility to create the database, `scheddb`, for example. See your Sybase product documentation for details.

Results

The Sybase database for the scheduler exists.

Scheduler table management functions:

The administration console and the WASSchedulerConfiguration MBeans provide simplified methods for creating scheduler tables and schema, verifying that the scheduler tables and schema are setup properly and are accessible and removing scheduler tables and schema.

Note: There are limitations when running the table management functions relating to data source access. See the Verifying a connection topic for details on these limitations. If a connection cannot be verified successfully, the scheduler table management functions will fail.

Verify tables

Validates that scheduler data sources, table prefixes, security authentication information and tables are configured correctly. You can use this verification method in production and development environments without altering database properties.

Create tables

Creates the necessary tables and indices required for schedulers to operate. This method of creating scheduler tables is designed for simple topologies and development environments. Use the supplied scheduler data definition language files for advanced or production environments and for databases that do not support this feature. For details, see the topic Creating scheduler tables using the administrative console.

Drop tables

Specifies the removal of tables and indices required for schedulers to operate. This method of removing scheduler tables and indices is recommended for development environments. When you drop tables, the action removes all previously scheduled tasks, and the scheduler no longer operates successfully, until the tables are recreated.

Scheduler table definition:

Schedulers require database tables and indices with a table prefix. This page provides reference information about the tables.

Each scheduler requires several database tables and indices to operate. Each table name and index described in this topic requires a table prefix. For example, if the scheduler is configured with a table prefix value, **SCHED_**, the table with the name, **TASK**, would be named **SCHED_TASK**. See Scheduler settings for details on the table prefix.

To create the tables, see Creating the database for schedulers. To see the exact schema definition such as field sizes and types, see Creating scheduler tables using DDL files. This section references the location where the DDL or SQL statements are stored. These statements create the table schema.

Note: The information in this topic is provided for problem determination. Do not alter the scheduler table names, field names or index names. The data content format might change without notice. Be aware of this factor when accessing the tables directly. Modifying data in the tables without using the Scheduler API might cause failures.

TASK

The TASK table contains the tasks that have been scheduled, but not yet purged. The primary key for this table is the TASKID which equates to the getTaskID() method on the com.ibm.websphere.scheduler.TaskStatus interface.

Since there is one row in this table for each task, it is important that the database and table support row-locking. Using page, or table locks, inhibits the scheduler from running tasks concurrently.

Field name	Purpose and notes
TASKID	Contains all of the tasks that have been scheduled, but not yet purged. The primary key for this table is TASKID which equates to the getTaskID() method on the com.ibm.websphere.scheduler.TaskStatus interface. Since there is one row in this table for each task, it is important that the database and table support row-locking. Using page, or table locks, will inhibit the scheduler from running tasks concurrently.
VERSION	Internal version ID of this row format.
ROW_VERSION	The version of this row. Used for optimistic locking.
TASKTYPE	The type of task: **1**=BeanTaskInfo, **2**=MessageTaskInfo
TASKSUSPENDED	This value indicates if the task is suspended or if it is running. The task is suspended if the value BITWISE AND 1 equals **1**. The task is running if the value BITWISE AND 2 equals **2**.
CANCELLED	The value, **1**, if the task is cancelled.
NEXTFIRETIME	The date in milliseconds using java.util.Date.getTime() when the task is scheduled to run next.
STARTBYINTERVAL	The start-by-interval of the task.
STARTBYTIME	Reserved.
VALIDFROMTIME	The task start time.
VALIDTOTIME	Reserved.
REPEATINTERVAL	The task repeat interval.
MAXREPEATS	The number of times to run the task.
REPEATSLEFT	The number of times the task has yet to run.
TASKINFO	Internal binary data.
NAME	The task name.
AUTOPURGE	The value, **1**, if the task is to automatically purge upon completion.
FAILUREACTION	Reserved.
MAXATTEMPTS	Reserved.
QOS	Reserved.
PARTITIONID	Reserved.
OWNERTOKEN	The task owner.
CREATETIME	The time in milliseconds using java.util.Date.getTime() when the task was created.

The TASK table also has the following indices that are required to allow the scheduler to run and access tasks concurrently:

- TASK_IDX1 – Used to access individual tasks using the Scheduler API.
- TASK_IDX2 – Used by the poll daemon to load expiring tasks.

TREG

The TREG table is used to store scheduler information that is shared between redundant schedulers. This table is not highly used.

Field name	Purpose and notes
REGKEY	The registry key. This is the primary key of the table.
REGVALUE	The registry value.

LMGR

The LMGR table is used to track the leases that redundant schedulers use. This table is not highly used.

Field name	Purpose and notes
LEASENAME	The name of the lease. This is the scheduler JNDI name and is the primary key.
LEASEOWNER	The owner of the lease. The format is `Cell/Node/Server`.
LEASE_EXPIRE_TIME	The time in milliseconds using java.util.Date.getTime() when the lease for the scheduler expires.
DISABLED	Reserved.

LMPR

The LMPR table is used to store arbitrary properties for the lease. This table is not highly utilized.

Field name	Purpose and notes
LEASENAME	The name of the lease. See the LMGR table.
NAME	The name of the property.
VALUE	The value of the property.

The LMPR table also has the following index:

- LMPR_IDX1 – Used to retrieve properties for a given lease.

Creating scheduler tables using the administrative console:

To create scheduler tables using the administrative console, the scheduler requires a database, a Java DataBase Connectivity (JDBC) provider and a data source.

Before you begin

Note: Limitations for Oracle XA databases

Oracle XA prohibits required schema operations in a global transaction environment. Local transactions are not supported. If you have schedulers that use an Oracle XA data source, either temporarily change the scheduler configuration to use a non-XA Oracle data source, or create the tables manually using the supplied DDL files. If you use the administrative console to create or drop scheduler tables for a scheduler configured to use an Oracle XA data source, then you receive a SchedulerDataStoreException error message, and the operation fails.

Note: Limitations for DB2 z/OS databases

Creating and dropping tables using the administrative console is not supported for DB2 z/OS databases. A database administrator is typically involved with defining and managing databases on DB2 z/OS systems. The administration interface is targeted for the non-database administrator or developer who does not want to know the specifics of setting up the scheduler database. The scheduler has DDL files available for the database administrator to create the required tables.

1. Verify that the database to be used for this scheduler is available and accessible by the application server. Review the Creating scheduler databases and tables topic for instructions on creating a database. The remaining steps describe how to create scheduler tables in an existing database.
2. Start the administrative console.
3. Create a JDBC data source that refers to the scheduler database.
4. Test the data source connection.
5. Create a scheduler. This configuration object contains the desired table prefix and the JNDI name of the JDBC data source. Verify that you save the new Scheduler to the configuration repository before you proceed to the next step.
6. Click **Resources** > **Schedulers** to view all defined schedulers.
7. Select one or more schedulers.
8. Click **Create Tables** to create the tables for the selected schedulers in their associated database. The tables and indices you created reflect the table prefixes and data sources specified in each scheduler configuration.
9. Restart the server or start the poll daemon to run scheduler tasks.

Results

Scheduler tables and schema are created.

Creating scheduler tables using scripting and Java Management Extensions:

Creating scheduler tables using scripting and Java Management Extensions requires a database, a Java DataBase Connectivity (JDBC) provider, and a data source.

Before you begin

Note: Limitations for Oracle XA databases

> Oracle XA prohibits required schema operations in a global transaction environment. Local transactions are not supported. If you have schedulers that use an Oracle XA data source, either temporarily change the scheduler configuration to use a non-XA Oracle data source, or create the tables manually using the supplied DDL files. If you use the administrative console to create or drop scheduler tables for a scheduler configured to use an Oracle XA data source, then you receive a SchedulerDataStoreException error message, and the operation fails.

Note: Limitations for DB2 z/OS databases

Creating and dropping tables using the administrative console is not supported for DB2 z/OS databases. A database administrator is typically involved with defining and managing databases on DB2 z/OS systems. The administration interface is targeted for the non-database administrator or developer who does not want to know the specifics of setting up the scheduler database. The scheduler has DDL files available for the database administrator to create the required tables.

1. Verify that the database to be used for this Scheduler is available and accessible by the application server. Review the Creating scheduler databases and tables topic for instructions on creating a database. The remainder of these steps describe how to create scheduler tables in an existing database.

2. Launch the wsadmin tool and connect to an application server. This process requires an active server to be available and fails, if you are disconnected from the server.
3. Create a JDBC data source that refers to the scheduler database.
4. Test the data source connection.
5. Create a scheduler. This configuration object contains the desired table prefix and the JNDI name of the JDBC data source. Verify that you save the new Scheduler to the configuration before you proceed to the next step.
6. Run the **createTables** MBean operation.
   a. Look up the SchedulerConfiguration object or use the object you created in a previous step.
   b. Locate the **WASSchedulerConfiguration** MBean.
   c. Run one of the **createTables** MBean operation on the **WASSchedulerConfiguration** object to create the tables for the specified **SchedulerConfiguration** object in its associated database. The tables and indices that you created reflect the table prefix and data source specified in the scheduler configuration.
7. Restart the server or start the poll daemon to run scheduler tasks.

Results

Scheduler tables and schema are created.

Example: Using scripting to verify scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and verify tables and indices for a scheduler.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which verifies that the tables and indices are created correctly for a scheduler. See the “Configuring Schedulers” topic for details on how a scheduler is created.

```
# Example JACL Script to verify the scheduler tables

# The name of the scheduler to verify
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler $schedName could not be found."
    puts ""
    exit
}

# Invoke the verifyTables method on the helper MBean.

puts ""
puts "Verifying tables for:"
puts "$myScheduler"
puts ""

if { [catch {$AdminControl invoke $schedHelper verifyTables $myScheduler} errorInfo] } {
    puts ""
    puts "Error verifying tables: $errorInfo"
    puts ""
} else {
```

```
    puts ""
    puts "Tables verified successfully."
    puts ""
}
```

Example: Using scripting to create scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and create scheduler tables.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which creates the scheduler tables for a configured scheduler. See the Configuring Schedulers topic for details on how to create a scheduler.

```
# Example JACL Script to create the scheduler tables

# The name of the scheduler to create tables for
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler with name: $schedName could not be found."
    puts ""
    exit
}

# Invoke the createTables method on the helper MBean.

puts ""
puts "Creating tables for:"
puts "$myScheduler"
puts ""

if {[catch {
    set result [$AdminControl invoke $schedHelper createTables $myScheduler]
    if {$result} {
        puts ""
        puts "Successfully created the tables."
        puts ""
    } else {
        puts ""
        puts "The tables were already created."
        puts ""
    }
  } errorInfo ] } {
    puts ""
    puts $errorInfo
    puts ""
}
```

Example: Using scripting to drop scheduler tables:

Use the wsadmin scripting tool to invoke a `Jacl` script and remove scheduler tables.

The following `Jacl` example script can be invoked using the wsadmin scripting tool, which removes the scheduler tables for a configured scheduler. See the “Configuring Schedulers” topic for details on how a scheduler is created

```
# Example JACL Script to drop the scheduler tables

# The name of the scheduler to drop the tables for
set schedName "My Scheduler"

puts ""
puts "Looking-up Scheduler Configuration Helper MBean"
puts ""
set schedHelper [$AdminControl queryNames WebSphere:*,type=WASSchedulerCfgHelper]

#Access the configuration object.
set myScheduler [$AdminConfig getid /SchedulerConfiguration:$schedName/]

if {$myScheduler == ""} {
    puts ""
    puts "Error: Scheduler with name: $schedName could not be found."
    puts ""
    exit
}

# Invoke the dropTables method on the helper MBean.

puts ""
puts "Dropping tables for:"
puts "$myScheduler"
puts ""

if {[catch {
    set result [$AdminControl invoke $schedHelper dropTables $myScheduler]
    if {$result} {
        puts ""
        puts "Successfully dropped the tables."
        puts ""
    } else {
        puts ""
        puts "The tables were already dropped."
        puts ""
    }
  } errorInfo ] } {
    puts ""
    puts $errorInfo
    puts ""
}
```

Creating scheduler tables using DDL files:

This topic provides the steps for creating scheduler tables using DDL files.

Before you begin

Your database system must be installed and available.

The scheduler requires a database, a Java™ Database Connectivity (JDBC) provider, and a data source.

About this task

Complete the following steps to create scheduler tables using DDL files.

1. Verify that your database is created. See the topic ″Creating scheduler databases.″
2. Create the database tables according to the instructions for your database system.
   - “Creating Apache Derby tables for schedulers” on page 2368
   - Creating DB2 tables for schedulers.
   - .

- Creating DB2 for iSeries tables for schedulers.
- Creating Informix tables for schedulers.
- Creating Microsoft SQL Server tables for schedulers.
- Creating Oracle tables for schedulers.
- Creating Sybase tables for schedulers.

The following scripts are deprecated, but will still work:

Deprecated script	New script
createSchemaDB2.ddl	createSchemaMod1DB2.ddl
createSchemaDB2iSeries.ddl	createSchemaMod1DB2iSeries.ddl
createSchemaDerby.ddl	createSchemaMod1Derby.ddl
createSchemaMSSQL.sql	createSchemaMod1MSSQL.sql
createSchemaInformix.sql	createSchemaMod1Informix.sql
createSchemaOracle.ddl	createSchemaMod1Oracle.ddl
createSchemaSybase12.ddl	createSchemaMod1Sybase12.ddl
dropSchemaMSSQL.sql	dropSchemaMod1MSSQL.sql
dropSchemaDB2.ddl	dropSchemaMod1DB2.ddl
dropSchemaDB2iSeries.ddl	dropSchemaMod1DB2iSeries.ddl
dropSchemaDerby.ddl	dropSchemaMod1Derby.ddl
dropSchemaInformix.sql	dropSchemaMod1Informix.sql
dropSchemaOracle.ddl	dropSchemaMod1Oracle.ddl
dropSchemaSybase12.ddl	dropSchemaMod1Sybase12.ddl

Creating Apache Derby tables for schedulers:

This topic describes how to create tables for schedulers on Apache Derby databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″Creating Cloudscape databases for schedulers″ topic, for more information.

About this task

To create tables for schedulers on Apache Derby databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Create the schema.
   a. Using a text editor, edit the script, *%app_server_root%*\Scheduler\createSchemaMod1Derby.ddl, according to the instructions at the top of the file.

      Note: When setting the table prefix, capitalize all characters.
   b. Enter one of the following commands.

      Note: Apache Derby provides both an embedded and network server version. This example is for the embedded version of Apache Derby. See the Apache Derby product documentation for

more details on running DDL scripts.

On Windows systems (using the example name, scheddb):

```
%app_server_root%\derby\bin\embedded\ij.bat %app_server_root%\Scheduler\createSchemaMod1Derby.ddl
```

On UNIX systems (using the example name, scheddb):

```
%app_server_root%/derby/bin/embedded/ij.sh %app_server_root%/Scheduler/createSchemaMod1Derby.ddl
```

Results

The Apache Derby tables and schema for the scheduler exist.

Creating DB2 tables for schedulers:

This topic describes how to create tables for scheduler on DB2 databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″Creating DB2 databases for schedulers″ topic for more information.

About this task

To create tables for scheduler on DB2 databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a DB2 command-line window.
2. Verify that you have administrator rights for the database system.
3. Create the table space and schema.
   a. Analyze the results of your experiences during development and system testing. The size of your database depends on many factors. If possible, distribute table space containers across different logical disks, and implement an appropriate security policy. Consider the performance implications of your choices for buffer pools and log file settings.
   b. Using a text editor, edit the following scripts according to the instruction at the top of each file.

      Note: When setting the table prefix, capitalize all characters.

      %WAS_HOME%\Scheduler\createTablespaceDB2.ddl, %WAS_HOME%\Scheduler\createSchemaMod1DB2.ddl, %WAS_HOME%\Scheduler\dropSchemaMod1DB2.ddl, and %WAS_HOME%\Scheduler\dropTablespaceDB2.ddl.
   c. Verify that you are attached to the correct instance. Check the environment variable DB2INSTANCE.
   d. To connect to the database, scheddb, for example, and enter the command:

      ```
      db2 connect to scheddb
      ```
   e. Create the table space. Enter the following command:

      ```
      db2 -tf createTablespaceDB2.ddl
      ```

      Verify that the script output contains no errors. If there were any errors, you can drop the table space using the following script:

      ```
      dropTablespaceDB2.ddl
      ```
   f. To create the schema (tables and indices), in the DB2 command line processor, enter the command db2 -tf createSchemaMod1DB2.ddl. Verify that the script output contains no errors. If there were any errors, you can use the following file to drop the schema:

      ```
      dropSchemaMod1DB2.ddl
      ```
   g. Verify that the DB2_RR_TO_RS DB2 flag is set to **YES** to avoid deadlocks. Restart the DB2 instance to activate the change, if needed.

Results

The DB2 tables and schema for the scheduler exist.

Creating DB2 for iSeries tables for schedulers:

This topic describes how to create tables for scheduler on DB2 for iSeries databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″ Creating DB2 for iSeries databases for schedulers″ topic for more information.

About this task

To create tables for scheduler on DB2 for iSeries databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Start a QShell session by running the following command from a CL command line:
   ```
   STRQSH
   ```
2. Create a new IFS directory under your profile directory:
   ```
   mkdir profile_root/scheduler
   ```
   Where *profile_root* is the fully qualified path of the directory that contains your profile.
3. Copy the following scripts to this new directory:
   ```
   %WAS_HOME%/scheduler/createDB2iSeriesSchema.ddl
   %WAS_HOME%/scheduler/dropSchemaDB2iSeries.ddl
   ```
4. Using a text editor, edit these scripts in the new directory. Replace all occurrences of @TABLE_PREFIX@ with <*collection_name*>, <*table_prefix*> where <*collection_name*> is the name of the collection that will be used to store the database files and is the collection that was created in the ″ Creating DB2 for iSeries databases for schedulers″ topic. The <*table_prefix*> precedes each table name.
5. Use the iSeries Navigator to create the database files on your iSeries server.
   a. Start iSeries Navigator.
   b. Expand the iSeries icon to locate the system where you want to create the database files.
   c. Expand **Database**, and right-click the system database.
   d. Select **Run SQL Scripts**.
   e. Select **File** > **Open**.
   f. Navigate to the `createDB2iSeriesSchema.ddl` file in your profile directory.
   g. Select **Run** > **All**.
   h. Select **View** > **Job Log** to verify that the tables were created.
6. Execute the following command to change the owner for the new database files to QEJBSVR:
   ```
   CHGOBJOWN OBJ(*ALL) OBJTYPE(*ALL) NEWOWN(QEJBSVR)
   ```

Results

The DB2 for iSeries tables and schema for the scheduler exist.

Creating Informix tables for schedulers:

This topic describes how to create tables for schedulers on Informix databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires that you configure a database and make it available. See the ″Creating Informix databases for schedulers″ topic for more information.

About this task

To create tables for schedulers on Informix databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Verify that you have administrator rights for the database system.
3. Create the schema.
   a. Using a text editor, edit the script `%WAS_HOME%\Scheduler\createSchemaMod1Informix.sql` according to the instruction at the top of the file.

      Note: When setting the table prefix, capitalize all characters.
   b. Enter the following command, using the database, `scheddb`, for example:

      ```
      dbaccess scheddb createSchemaMod1Informix.sql
      ```

Results

The Informix tables and schema for the scheduler exist.

Creating Microsoft SQL Server tables for schedulers:

This topic describes how to create tables for schedulers on Microsoft SQL Server databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″Creating Microsoft SQL Server databases for schedulers″ topic for more information.

About this task

To create tables for schedulers on Microsoft SQL Server databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Change to the directory where the configuration scripts for scheduler are located. This is the Scheduler subdirectory of the IBM WebSphere Application Server installation directory.
3. Use a text editor to edit the schema creation script, `createSchemaMod1MSSQL.sql`, according to the instructions at the beginning of the file.

   Note: When setting the table prefix, capitalize all characters.
4. Create the schema:
   a. Verify that you have administrator rights for the database system. The user ID you use to create the schema must be the one that you configure WebSphere Application Server to use when accessing the database.
   b. Run the following script to create the schema (tables and views) :

      ```
      isql -S <servername> -U<userid> -P<password> -D<databaseName> -i<script name>
      ```

Results

The Microsoft SQL Server tables and schema for scheduler exist.

Creating Oracle tables for schedulers:

This topic describes how to create tables for schedulers on Oracle databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″Creating Oracle databases for schedulers″ topic for more information.

About this task

To create tables for schedulers on Oracle databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.
3. Create the table space and schema.
   a. Using a text editor, edit the following scripts according to the instructions at the top of the files.

      Note: When setting the table prefix, capitalize all characters.

      %app_server_root%\Scheduler\createTablespaceOracle.ddl and *%app_server_root%*\createSchemaMod1Oracle.ddl
   b. Set the environment variable ORACLE_SID, if you do not want the schema to be created in the default instance.
   c. Run the script, createTablespaceOracle.ddl, to create the table space.

      For test purposes, use the same location for all table spaces and pass the path as a command line argument to the script.
   d. Run the script, createSchemaMod1Oracle.ddl, to create the schema.

Results

The Oracle tables and schema for scheduler exist.

Creating Sybase tables for schedulers:

This topic describes how to create tables for schedulers on Sybase databases, using data definition language (DDL) or structured query language (SQL) files.

Before you begin

This task requires you to configure a database and make it available. See the ″Creating Sybase databases for schedulers″ topic for more information.

About this task

To create tables for schedulers on Sybase databases, using data definition language (DDL) or structured query language (SQL) files, use these steps.

1. Open a command-line window.
2. Make sure that you have administrator rights for the database system.

3. Make sure that you have the Distributed Transaction Management (DTM) option for Sybase ASE installed.
   a. Set enable DTM to **1** in the Sybase server configuration.
   b. Set enable xact coordination to **1** in the Sybase server configuration.
   c. Add the **dtm_tm_role** role to the Sybase administration user ID. For example, enter the user ID `sa`.
   d. Restart the Sybase server.
4. Use the Sybase isql utility to create a database. For example, enter the database name `scheddb`. See your Sybase product documentation for details.
5. Create the schema:
   a. Using a text editor, edit the following script according to the instructions located at the top of the file.

      Note: When setting the table prefix, capitalize all characters.

      app_server_root`\Scheduler\createSchemaMod1Sybase12.ddl`
   b. Enter the following command:

      ```
      isql -S <servername> -U <userid> -P <password> -D scheddb -i createSchemaMod1Sybase12.ddl
      ```

Results

The Sybase tables and schema for scheduler exist.

Startup beans

Using startup beans

There are two types of startup beans: application startup beans and Module startup beans.

About this task

A module startup bean is a session bean that is loaded when an EJB Jar file starts. Module startup beans enable Java Platform Enterprise Edition (Java EE) applications to run business logic automatically, whenever an EJB module starts or stops normally. An application startup bean is a session bean that is loaded when an application starts. Application startup beans enable Java EE applications to run business logic automatically, whenever an application starts or stops normally.

Startup beans are especially useful when used with asynchronous bean features. For example, a startup bean might create an alarm object that uses the Java Message Service (JMS) to periodically publish heartbeat messages on a well-known topic. This enables clients or other server applications to determine whether the application is available. Refer to the Enabling an application to wait for a messaging engine to start article if you are using the default JMS provider.

1. For Application startup beans, use the home interface, com.ibm.websphere.startupservice.AppStartUpHome, to designate a bean as an Application startup bean. For Module startup beans, use the home interface, com.ibm.websphere.startupservice.ModStartUpHome, to designate a bean as a Module startup bean.
2. For Application startup beans, use the remote interface, com.ibm.websphere.startupservice.AppStartUp, to define start() and stop() methods on the bean. For Module startup beans, use the remote interface, com.ibm.websphere.startupservice.ModStartUp, to define start() and stop() methods on the bean.

   The startup bean start() method is called when the module or application starts and contains business logic to be run at module or application start time.

The start() method returns a boolean value. **True** indicates that the business logic within the start() method ran successfully. Conversely, **False** indicates that the business logic within the start() method failed to run completely. A return value of False also indicates to the Application server that application startup is aborted.

The startup bean stop() methods are called when the module or application stops and contains business logic to be run at module or application stop time. Any exception thrown by a stop() method is logged only. No other action is taken.

The start() and stop() methods must never use the TX_MANDATORY transaction attribute. A global transaction does not exist on the thread when the start() or stop() methods are invoked. Any other TX_* attribute can be used. If TX_MANDATORY is used, an exception is logged, and the application start is aborted.

The start() and stop() methods on the remote interface use **Run-As** mode. **Run-As** mode specifies the credential information to be used by the security service to determine the permissions that a principal has on various resources. If security is on, the **Run-As** mode needs to be defined on all of the methods called. The identity of the bean without this setting is undefined.

There are no restrictions on what code the start() and stop() methods can run, since the full Application Server programming model is available to these methods.

3. Use an *optional* environment property integer, `wasStartupPriority`, to specify the start order of multiple startup beans in the same Java Archive (JAR) file. If the environment property is found and is the wrong type, application startup is aborted. If no priority value is specified, a default priority of `0` is used. It is recommended that you specify the priority property. Beans that have specified a priority are sorted using this property. Beans with numerically lower priorities are run first. Beans that have the same priority are run in an undefined order. All priorities must be positive integers. Beans are stopped in the opposite order to their start priority. The priority values for module startup beans and application startup beans are mutually exclusive. All modules will be started prior to the application being declared as ″started″ and therefore the start() methods for module startup beans within an application will be invoked prior to the start() methods for any application startup beans. Likewise, all application startup bean stop() methods for a specific Java Archive (JAR) file will be invoked prior to any module startup bean stop() methods for that JAR.
4. For module startup beans, the order in which EJB modules are started can be adjusted via the ″Starting weight″ value associated with each module
5. To control who can invoke startup bean methods via WebSphere Security do the following:
   a. Define the method permissions for the Start() and Stop() methods as you would for any EJB module. (See ″Defining method permissions for EJB modules″.)
   b. Ensure that the user that is mapped to the Security Role defined for the startup bean methods is the same user that is defined as the Server user ID within the User Registry.

What to do next

View the startup beans service settings.

Startup beans service settings

Use this page to enable startup beans that control whether application-defined startup beans function on this server. Startup beans are session beans that run business logic through the invocation of start and stop methods when applications start and stop. If the startup beans service is disabled, then the automatic invocation of the start and stop methods does not occur for deployed startup beans when the parent application starts or stops. This service is disabled by default. Enable this service only when you want to use startup beans. Startup beans are especially useful when used with asynchronous beans.

To view this administrative console page, click **Servers** > **Application servers** >*server_name* > **Container services** > **Startup beans service**.

Enable service at server startup:

Specifies whether the server attempts to initiate the startup beans service.

Default
Cleared

Range

Selected
When the application server starts, it attempts to initiate the startup bean service automatically.

Cleared
The server does not try to initiate the startup beans service. All startup beans do not start or stop with the application. If you use startup beans on this server, then the system administrator must start the startup beans service manually or select this property, and then restart the server.

Enabling startup beans in the administrative console

Enabling startup beans in the administrative console enables Java 2 Platform Enterprise Edition (J2EE) applications to run business logic automatically, whenever an application starts or stops normally.

About this task

Use the following steps to enable startup beans in the administrative console.

1. Start the administrative console.
2. Select **Servers** > **Application Servers** > *server_name* > **Container Services** > **Startup beans service**.
3. Select the **Enable service at server startup** check box.
4. Click **Apply** to save the configuration.

What to do next

View the startup beans service settings.

Work area

Task overview: Implementing shared work areas

About this task

The work area service enables application developers to implicitly propagate information beyond the information passed in remote calls. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of each method. The methods on the server side can use or ignore the information in the work area as appropriate.

Before proceeding with the steps to implement work areas, as described below, review the topic Work area service: Overview.

1. Developing applications that use work areas. Applications interact with the work area service by implementing the UserWorkArea interface.
2. Managing work areas. The work area service is managed using the administrative console.
3. “Configuring work area partitions” on page 2384 You can create multiple work areas with additional configuration options than the UserWorkArea partition.
4. “Propagating work area context over Web services” on page 2394 You can propagate work area context on a Web service call instead of an RMI/IIOP call.

Overview of work area service

The work area service passes information explicitly as an argument or implicitly to remote methods.

One of the foundations of distributed computing is the ability to pass information, typically in the form of arguments to remote methods, from one process to another. When application-level software is written over middleware services, many of the services rely on information beyond that passed in the application's remote calls. Such services often make use of the implicit propagation of private information in addition to the arguments passed in remote requests; two typical users of such a feature are security and transaction services. Security certificates or transaction contexts are passed without the knowledge or intervention of the user or application developer. The implicit propagation of such information means that application developers do not have to manually pass the information in method invocations, which makes development less error-prone, and the services requiring the information do not have to expose it to application developers. Information such as security credentials can remain secret.

The work area service gives application developers a similar facility. Applications can create a work area, insert information into it, and make remote invocations. The work area is propagated with each remote method invocation, eliminating the need to explicitly include an appropriate argument in the definition of every method. The methods on the server side can use or ignore the information in the work area as appropriate. If methods in a server receive a work area from a client and subsequently invoke other remote methods, the work area is transparently propagated with the remote requests. When the creating application is done with the work area, it terminates it.

There are two prime considerations in deciding whether to pass information explicitly as an argument or implicitly by using a work area. These considerations are:
- Pervasiveness: Is the information used in a majority of the methods in an application?
- Size: Is it reasonable to send the information even when it is not used?

When information is sufficiently pervasive that it is easiest and most efficient to make it available everywhere, application programmers can use the work area service to simplify programming and maintenance of code. The argument does not need to go onto every argument list. It is much easier to put the value into a work area and propagate it automatically. This is especially true for methods that simply pass the value on but do nothing with it. Methods that make no use of the propagated information simply ignore it.

Work areas can hold any kind of information, and they can hold an arbitrary number of individual pieces of data, each stored as a property.

Use the work area service in the administrative console to configure the UserWorkArea partition. The UserWorkArea partition is the partition that is available in JNDI naming under the name ″java:comp/websphere/UserWorkArea″ as demonstrated in the article, "Accessing the UserWorkArea partition" on page 2383. The UserWorkArea partition is the default work area partition created automatically, if it has not been disabled, and is available through JNDI naming to all users. Any configuration option made to the UserWorkArea partition under the work area service panel in the administrative console does not affect the work area partition service or any partitions defined in it, and conversely. For example, if you select the enable or disable option in the work area service panel, this does not affect the work area partition service or any partition within it.

Work area property modes:

The information in a work area consists of a set of properties; a property consists of a key-value-mode triple. The key-value pair represents the information contained in the property; the key is a name by which the associated value is retrieved. The mode determines whether you can modify or remove the property.

Property modes

There are four possible mode values for properties, as shown in the following code example:

Code example: The PropertyModeType definition

```
public final class PropertyModeType {
   public static final PropertyModeType normal;
   public static final PropertyModeType read_only;
   public static final PropertyModeType fixed_normal;
   public static final PropertyModeType fixed_readonly;
};
```

A property's mode determines three things:
- Whether the value associated with the key can be modified
- Whether the property can be deleted
- Whether the mode associated with the key-value pair can be modified

The two read-only modes forbid changes to the information in the property; the two fixed modes forbid deletion of the property.

The work area service does not provide methods specifically for the purpose of modifying the value of a key or the mode associated with a property. To change information in a property, applications simply rewrite the information in the property; this has the same effect as updating the information in the property. The mode of a property governs the changes that can be made. Modifying key-value pairs describes the restrictions each mode places on modifying the value and deleting the property. Changing modes describes the restrictions on changing the mode.

Changing modes

The mode associated with a property can be changed only according to the restrictions of the original mode. The read-only and fixed read-only properties do not permit modification of the value or the mode. The fixed normal and fixed read-only modes do not allow the property to be deleted. This set of restrictions leads to the following permissible ways to change the mode of a property within the lifetime of a work area:
- If the current mode is normal, it can be changed to any of the other three modes: fixed normal, read-only, fixed read-only.
- If the current mode is fixed normal, it can be changed only to fixed read-only.
- If the current mode is read-only, it can be changed only by deleting the property and re-creating it with the desired mode.
- If the current mode is fixed read-only, it cannot be changed.
- If the current mode is not normal, it cannot be changed to normal. If a property is set as fixed normal and then reset as normal, the value is updated but the mode remains fixed normal. If a property is set as fixed normal and then reset as either read-only or fixed read-only, the value is updated and the mode is changed to fixed read-only.

Note: The key, value, and mode of any property can be effectively changed by terminating (completing) the work area in which the property was created and creating a new work area. Applications can then insert new properties into the work area. This is not precisely the same as changing the value in the original work area, but some applications can use it as an equivalent mechanism.

Nested work areas:

Applications can nest work areas to define and scope properties for specific tasks without having to make the work areas available to all parts of the application.

When an application creates a work area, a work area context is associated with the creating thread. If the application thread creates another work area, the new work area is nested within the existing work area

and becomes the current work area. All properties defined in the original, enclosing work area are visible to the nested work area. The application can set additional properties within the nested work area that are not part of the enclosing work area.

An application working with a nested work area does not actually see the nesting of enclosing work areas. The current work area appears as a flat set of properties that includes those from enclosing work areas. In the figure below, the enclosing work area holds several properties and the nested work area holds additional properties. From the outermost work area, the properties set in the nested work area are not visible. From the nested work area, the properties in both work areas are visible.

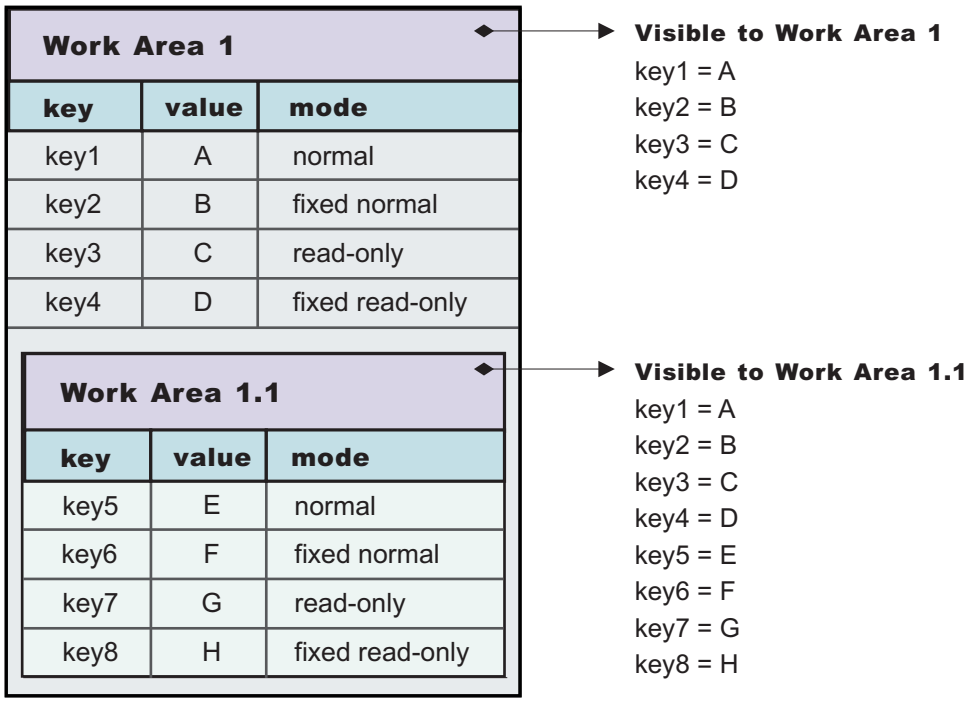


Figure 24. Defining new properties in nested work areas

Nesting can also affect the apparent settings of the properties. Properties can be deleted from or directly modified only within the work areas in which they were set, but nested work areas can also be used to temporarily override information in the property without having to modify the property. Depending on the modes associated with the properties in the enclosing work area, the modes and the values of keys in the enclosing work area can be overridden within the nested work area.

The mode associated with a property when it is created determines whether nested work areas can override the property. From the perspective of a nested work area, the property modes used in enclosing work areas can be grouped as follows:

- Modes that permit a nested work area to override the mode or the value of a key locally. The modes that permit overriding are:
  - Normal
  - Fixed normal
- Modes that do not permit a nested work area to override the mode or the value of a key locally. The modes that do not permit overriding are:
  - Read-only
  - Fixed read-only

If an enclosing work area defines a property with one of the modes that can be overridden, a nested work area can specify a new value for the key or a new mode for the property. The new value or mode becomes the value or mode seen by subsequently nested work areas. Changes to the mode are governed

by the restrictions described in Changing modes. If an enclosing work area defines a property with one of the modes that cannot be overridden, no nested work area can specify a new value for the key.

A nested work area can delete properties from enclosing work areas, but the changes persist only for the duration of the nested work area. When the nested work area is completed, any properties that were added in the nested area vanish and any properties that were deleted from the nested area are restored.

The following figure illustrates the overriding of properties from an enclosing work area. The nested work area redefines two of the properties set in the enclosing work area. The other two cannot be overridden. The nested work area also defines two new properties. From the outermost work area, the properties set or redefined in the nested work are not visible. From the nested work area, the properties in both work areas are visible, but the values seen for the redefined properties are those set in the nested work area.

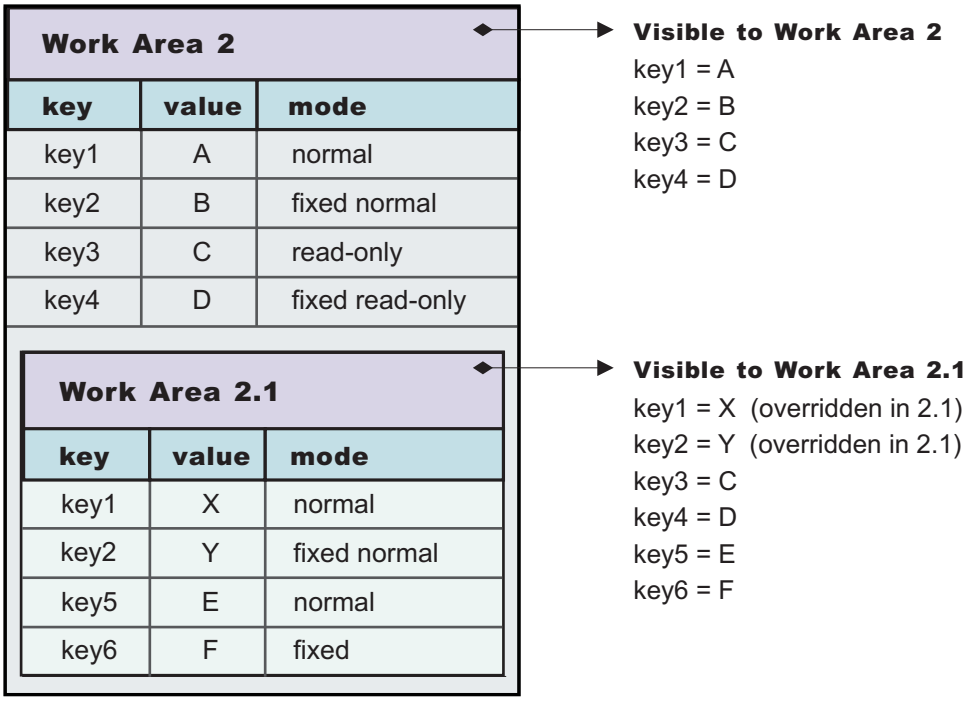


Figure 25. Redefining existing properties in nested work areas

Distributed work areas:

Work area context propagates to a target object on a remote invocation on both bidirectional and non-bidirectional defined work area partitions. The propagation of work area context operates differently depending on whether a work area partition is defined as bidirectional. If the partition is defined as bidirectional, the context propagates from a target object back to the originator.

Non-bidirectional work area partitions (UserWorkArea partition)

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects it invokes. However, no changes made to a nested work area on a target object are propagated back to the calling object. The caller's work area is unaffected by changes made in the remote method.

Bidirectional work area partitions

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors is propagated to the target. The target application can locally modify the information, as allowed by the property modes, this information is propagated to any remote objects it invokes. In a partition that is not defined as bidirectional, a target application must begin a nested work area before making changes to the imported work area. However, if a partition is defined as bidirectional, a target application need not begin a nested work area before operating on an imported work area. By not beginning a nested work area, any new context set into the work area, or any context changes made by the target application, is not only propagated on future remote invocations but is also propagated back to the originating application (that is, the one who initiated the remote invocation) thus allowing bidirectional propagation of work area context. If the target application does not want new or changed context to propagate back to the originating application, then the target application must begin a nested work area to scope the context to its process. However, the new or changed context in the nested work area propagates on any future remote invocation the target application may make.

WorkArea service: Special considerations:

Developers who use work areas should consider the following issues that could potentially cause problems: interoperability between the EJB and CORBA programming models; and the use of work areas with Java's Abstract Windowing Toolkit.

EJB and CORBA interoperability

Although the work area service can be used across the EJB and CORBA programming models, many composed data types cannot be successfully used across those boundaries. For example, if a SimpleSampleCompany instance is passed from the WebSphere environment into a CORBA environment, the CORBA application can retrieve the SimpleSampleCompany object encapsulated within a CORBA Any object from the work area, but it cannot extract the value from it. Likewise, an IDL-defined struct defined within a CORBA application and set into a work area is not readable by an application using the UserWorkArea class.

Note: Applications can avoid this incompatibility by directly setting only primitive types, like integers and strings, as values in work areas, or by implementing complex values with structures designed to be compatible, like CORBA valuetypes.

Also, CORBA Anys that contains either the tk_null or tk_void typecode can be set into the work area by using the CORBA interface. However, the work area specification cannot allow the Java 2 Platform, Enterprise Edition (J2EE) implementation to return null on a lookup that retrieves these CORBA-set properties without incorrectly implying that there is no value set for the corresponding key. For example, when a user attempts to retrieve a nonexistent key from a work area, the work area service returns null to indicate that the specified key does not contain a value, implying that the key itself is not in use or does not exist. In the case where CORBA Anys contains either tk_null or tk_void, when a user requests the key associated with one of these values, the work area service returns null as expected. In this case, the key may actually exist and the work area service was simply returning the key's value of null. Therefore, when working with CORBA Anys, a user must not make any implications when a null is returned from a work area because it could mean that either there isn't a property associated with the given key, or that there is a property associated with the given key and it contains a tk_null or tk_void, for example, a null in the J2EE environment. If a J2EE application tries to retrieve CORBA-set properties that are non-serializable, or contain CORBA nulls or void references, the com.ibm.websphere.workarea.IncompatibleValue exception is raised.

Using work areas with Java's Abstract Windowing Toolkit (AWT)

Work areas must be used cautiously in applications that use Java's Abstract Windowing Toolkit (AWT). The AWT implementation is multithreaded, and work areas begun on one thread are not available on another. For example, if a program begins a work area in response to an AWT event, such as pressing a button, the work area might not be available to any other part of the application after the execution of the event completes.

Work area service performance considerations: The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A *work area* is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, ″Deferred Attribute Serialization″. This mechanism attempts to minimize the number of serialization and deserialization calls. See “Work area partition service” on page 2384 for further explanation of this configuration attribute.

The maxSendSize and maxReceiveSize configuration parameters can affect the performance of the work area. Setting these two values to 0 (zero) effectively turns off the policing of the size of context that can be sent in a work area. This action can enhance performance, depending on the number of nested work areas an application uses. In applications that use only one work area, the performance enhancement might be negligible. In applications that have a large number of nested work areas, there might be a performance enhancement. However, a user must note that by turning off this policing it is possible that an extremely large amount of data might be sent to a server.

Performance is degraded if you use a work area as a direct replacement to passing a single parameter over a single method call. The reason is that you incur more overhead than just passing that parameter between method calls. Although the degradation is usually within acceptable tolerances and scales similarly to passing parameters with regard to object size, consider degradation a potential problem before utilizing the service. As with most functional services, intelligent use of the work areas yields the best results.

The work area service is a tool to simplify the job of passing information from resource to resource, and in some cases can improve performance by reducing the overhead that is associated with a parameter passing when the information is only sparsely accessed within the call graph. Caching the instance retrieved from JNDI is important to effectively maximize performance during runtime.

Managing the UserWorkArea partition

Before you begin

For an application to take advantage of work areas, the work area service must be enabled for both clients and servers. On a server the service is disabled by default. On the client, the service is enabled by default.

For an application to take advantage of the default partition, the UserWorkArea partition, this partition must be enabled by enabling the work area service for both clients and servers. The work area service on a server is disabled by default and the work area service on a client is enabled by default. Note that rather than using this default work area partition, a user can create their own work area partition using the "Work area partition service" on page 2384.

About this task

Applications can set maximum sizes on each work area that is sent or received. By default, the maximum size of a work area that is sent by a client and received, then possibly resent, by a server is 32,768 bytes. The maximum size that you can specify is determined by the maximum value expressible in the Java Integer data type, 2,147,483,647. The smallest maximum size that you can specify is 1. Using a maximum size of 1 byte effectively means that no requests associated with the work area can leave the system or enter another system. A value of 0 means that no limit is imposed. A value of -1 means that the default value is to be honored. The default value is also used if an invalid value or a malformed property is specified. You can change this size as described in this topic.

1. Enable or disable the use of the UserWorkArea partition on a server: The work area service is disabled by default on servers but enabled by default on the client
   a. Start the administrative console.
   b. Select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business Process Services** → **Work area service**.
   c. Select or clear the **Startup** check box. This specifies whether or not the server should automatically start the work area service when the server starts.
   d. Save the new configuration and restart the server to apply the new configuration.
2. Enable (or disable) the UserWorkArea partition on a client: Set the com.ibm.websphere.workarea.enabled property to TRUE or FALSE before starting the client. For example, to disable the work area service, when invoking the launchClient script found in the *app_server_root*/`bin` directory, add the following system property to the launchClient invocation:

   ```
   -CCDcom.ibm.websphere.workarea.enabled=false
   ```

   Alternatively, this property can be set in a property file that is used by the launchClient script. See "Running application clients" on page 203 for additional information.
3. Manage the size of the work areas that this server can send and the number of work areas that this server can accept.
   a. Start the administrative console.
   b. Select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business Process Services** → **Web container**.
      - To change the send size or receive size on the work area service (namely the ″UserWorkArea″ partition):
        – Select **Work area service**.
      - To change the send size or receive size on a user defined partition:
        – Select **Work area partition service**.
        – Select a partition.

c. Enter a new value in the **Maximum send size** field to modify the size of the work area that this server can send, or enter a new value in the **Maximum receive size** field to modify the size of the work area that this server can accept.

d. Save the new configuration and restart the server to apply the new configuration.

4. Change the size of the work area that can be sent by a client. This step only applies to the UserWorkArea partition on the client. To set the maximum send or receive size on a user defined partition, you must set these values when creating the partition on the client. For more information on creating a partition on a client, see the client section in the Configuring work area partitions topic. To change the size of the work area that can be sent by a client, set the com.ibm.websphere.workarea.maxSendSize property to the desired number of bytes before starting the client. You can set the maximum send size as follows:
   - Set the maximum send size when invoking the launchClient invocation script found in the `$WAS_HOME/bin` directory. For example, to set the maximum size to 10,000 bytes, add the following system properties to the launchClient invocation as needed: `-CCDcom.ibm.websphere.workarea.maxSendSize=10000`
   - Set the maximum send size propery, com.ibm.websphere.workarea.maxSendSize, in a property file that is used by the launchClient script. See ″“Running application clients” on page 203″ for additional information.

   Because the UserWorkArea partition is defined as unidirectional, for example, context only propagates on outbound calls and not on the return of those calls, the maximum receive size is ignored.

Accessing the UserWorkArea partition

About this task

The work area service provides a JNDI binding to an implementation of the UserWorkArea interface under the name java:comp/websphere/UserWorkArea. This is the default work area partition, namly the ″UserWorkArea″ partition. It is created and bound into JNDI naming automatically, as long as it is enabled as defined in Enabling the work area service (UserWorkArea partition). Applications that need to access UserWorkArea partition can perform a lookup on that JNDI name, as shown in the following code example:

Example

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

  InitialContext jndi = null;
  UserWorkArea userWorkArea = null;
  try {
     jndi = new InitialContext();
     userWorkArea = (UserWorkArea)jndi.lookup(
        "java:comp/websphere/UserWorkArea");
  }
  catch (NamingException e) { ... }
}
```

Rather than using this default work area partition, a user has the option to create their own work area partition using the Work area partition service.

What to do next

The next step is to use the begin method to create a new work area and associate it with the calling thread, as described in the topic, Beginning a new work area.

Configuring work area partitions

About this task

The work area partition service extends the work area service by allowing the creation of multiple work areas with more configuration options than what is available to the UserWorkArea partition. Follow these steps to create and configure a work area partition:

1. Create a user defined partition on the server.
   a. Start the administrative console.
   b. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business process services** → **Work area partition service**.
   c. Click **New**.
   d. On the settings page for work area partitions, specify values such as the partition name, maxSendSize and maxReceiveSize, then click OK.
   e. Save the new configuration.
   f. Restart the server to apply the new configuration.
2. Create a user defined partition on the client. Use the createWorkAreaPartition method described in the "The Work area partition manager interface" on page 2388 article to programmatically create a partition. See "Example: Using the work area partition manager" on page 2390 for an example of using this method.

Results

You have created a work area partition.

What to do next

Retrieve the partition through the work area partition manager interface and use it as defined by the work area service and the UserWorkArea interface. See the topic "Example: Using the work area partition manager" on page 2390 for an example.

Work area partition service

The work area partition service is an extension of the work area service that allows the creation of multiple custom work areas. The work area partition service is an optional service to users. Any user that currently uses the work area service and the UserWorkArea partition can continue using it in the same manner. The UserWorkArea partition is created automatically (if it has not been disabled) by the work area partition service. By allowing a user the option to create their own work area partition through the work area partition service, they can have more control over configuration and access to their partition.

The work area partition service is essentially a ″factory″ for creating instances of the UserWorkArea interface. Applications interact with work areas by using the UserWorkArea interface and its implementation. This interface defines all of the methods used to create, manipulate, and complete work areas. The work area partition service allows users to create their own named instance of the UserWorkArea interface, each named instance is called a user defined work area partition, or partition for short. Each instance of the UserWorkArea interface (partition) is separate from other user defined partitions. Furthermore, a partition can be configured with various configuration options to provide qualities of service unique to an individual user's use case. Any configuration option made within the work area partition service panel will not effect the work area service.

Unlike the UserWorkArea partition, which is publicly known, work areas created by the work area partition service are accessible to, and known only by the creator. However, the work area partition service does not strictly enforce that a partition is accessed and/or operated on exclusively by the partition creator. There are no limitations should the creator want to publish their work area partition and make it publicly available by binding their partition reference in java naming or by other means. However, the work area partition service does try to hide a partition as much as possible should a user not want others to know

about a certain partition. The work area partition service does not allow a person to determine, or query the names of all the partitions that have been created; however, it does not restrict the partitions from being accessed by users other than the creator of that partition. The context of a partition, such as the UserWorkArea partition or a user defined partition, is scoped to a single thread and is not accessible by multiple threads.

The work area partition reference that is returned to a user implements javax.naming. Referenceable, as well as com.ibm.websphere.UserWorkArea, therefore a user can bind their partition into a name to make their partition publicly available. An alternative to using Java naming to bind and access the partition is to use the work area partition manager interface. Anyone can access the work area partition manager interface; therefore, if a user wants to make their partition publicly available, they simply need to publish their partition name. Other users can then call the getWorkAreaPartition method on the work area partition manager interface with the published name.

The WorkAreaPartitionManager.createWorkAreaPartition method can only be used from a Java 2 Platform, Enterprise Edition (J2EE) client. To create a work area partition on the server side, one must use the administrative console. On the server side a work area partition must be created during server startup because each partition needs to be register with the appropriate Web and Enterprise JavaBeans (EJB) collaborators before the server has started. Custom work area partitions are created by the work area partition service and defined by the UserWorkArea interface.

The work area partition service also allows a user to configure partitions with additional properties that are not available on the UserWorkArea partition, such as bidirectional propagation of work area partition context and deferred attribute serialization. These properties are available as configuration properties when creating a partition. For a complete list of the configuration properties available when creating a partition, please see the section ″Configurable Work Area Partition Properties″ in “The Work area partition manager interface” on page 2388. The properties are defined as follows:

Bidirectional propagation of work area context

If a remote invocation is issued from a thread associated with a work area, a copy of the work area is automatically propagated to the target object, which can use or ignore the information in the work area as necessary. If the calling application has a nested work area associated with it, a copy of the nested work area and all its ancestors are propagated to the target application. The target application can locally modify the information, as allowed by the property modes, by creating additional nested work areas; this information is propagated to any remote objects that it invokes.

Whether context changes propagate back to a calling application from a remote application depends on the configuration of the work area partition. If a user creates a partition to be bidirectional (selects the Bidirectional property during creation), changes made by a remote application propagates back to the calling application, meaning that changes made to the work area context by a downstream process will propagate back up stream. The UserWorkArea partition is not configured (and can never be configured) to be bidirectional; therefore context changes only flow to downstream processes and do not propagate back upstream.

Example: Bidirectional propagation of work area context

Whether context changes propagate back to a calling application from a remote application depends on the configuration of the work area partition. If a user creates a bidirectional partition, changes made by a remote application propagate back to the calling application. Changes made to the work area context by a downstream process propagate back up stream. Figure 1 illustrates distribution of work area context on a remote call when the partition containing the given work area is configured for bidirectional propagation of its work area context. For this illustration, the client and server must have created a partition with the same name.

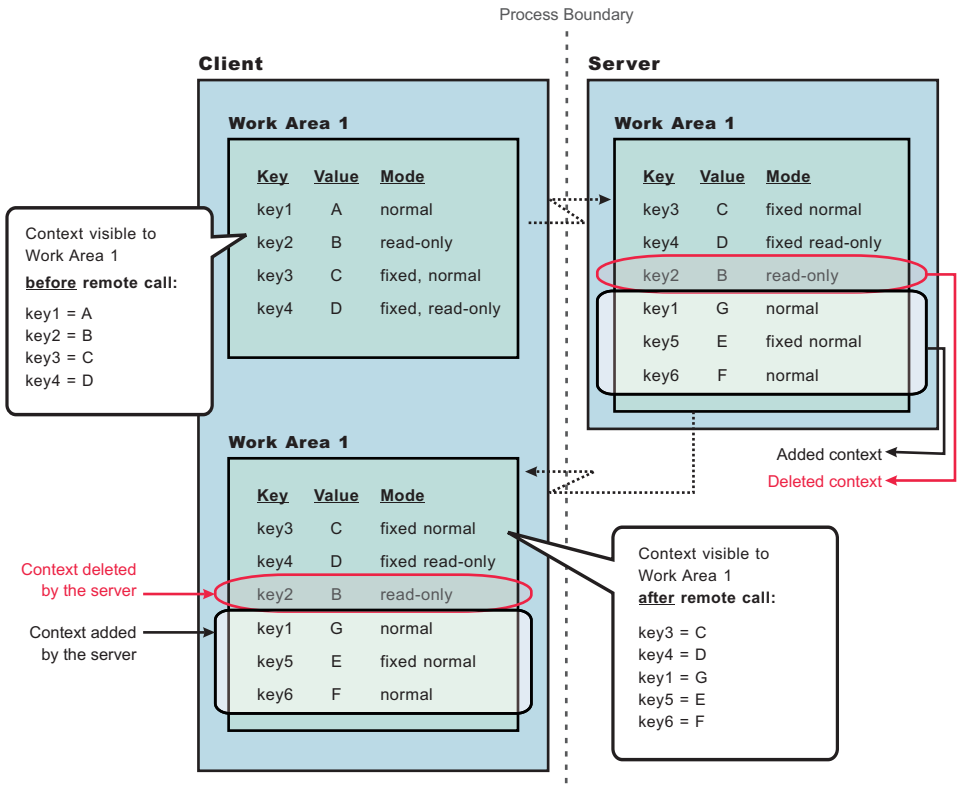


Figure 26. Figure 1

As Figure 1 shows, when the client makes a remote call to the server, the server receives the context set by the client process. The server then can make changes to this context or add to it. In this illustration, the server overwrites the value at **key1**, removes the property at **key2**, and adds two new properties at **key5** and **key6**. When the server application returns to the client, the work area context is propagated back to the client and demarshalled. The current work area is then updated with the new context. Note, that if the partition is not configured as bidirectional, and the server tries to change or remove context in work area, ″Work Area 1″, it receives a com.ibm.websphere.workarea.NotOriginator exception because the client was the originator of the work area. The server can retrieve the context in ″Work Area 1″. This is the main distinction between bidirectional propagation of context and non-bidirectional propagation.

Example: Bidirectional propagation of nested work area context

If a remote application needs to add context to a work area that is only used by itself or any other remote objects, the remote application must begin another work area. By beginning a new work area, the new context added is scoped to that application and does not flow back to the calling application. The major benefit of nesting work areas is that nesting work areas allows an application to scope work area context to a given application. Taking the above illustration one step further, if the server has begun a work area before overwriting the value at **key1**, removing the property at **key2**, or adding new properties at **key5** and **key6**; those changes would not have propagated back to the client. This is shown in Figure 2. You can also see from this figure that the client does not receive the context from the nested work area started by the server.

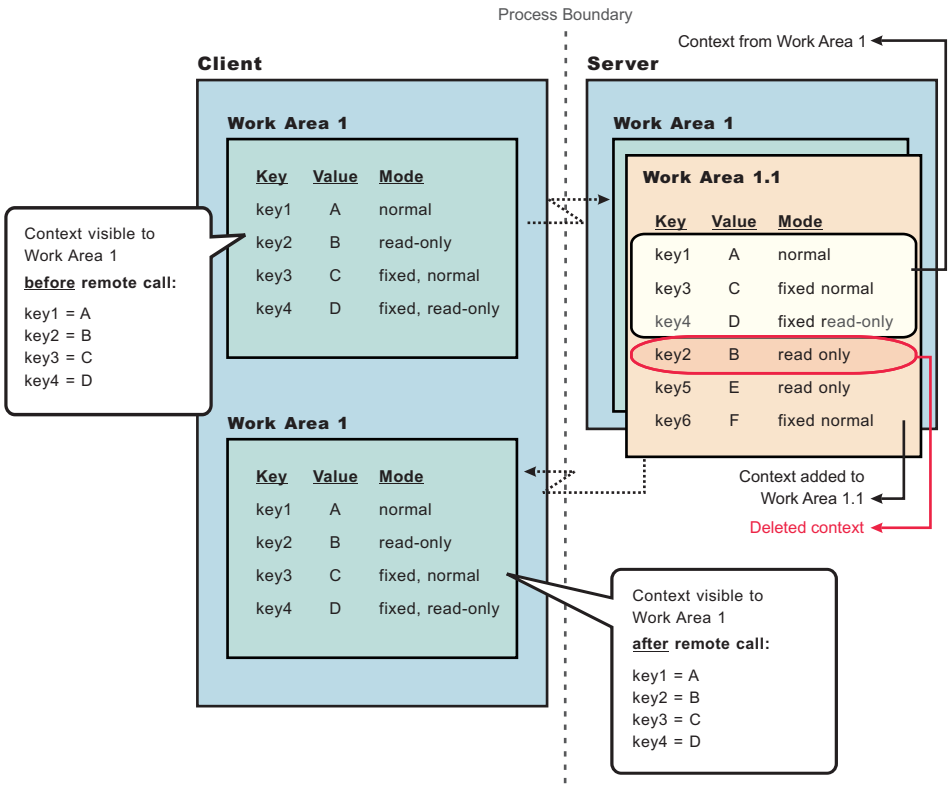


Figure 27. Figure 2

Deferred attribute serialization of work area context

By default, on each set operation the attribute set into a work area is automatically serialized by the work area service. On each subsequent get operation on that same attribute it is deserialized and returned to the requester. This gives the work area service complete control of the attribute such that any changes to a mutable object are not reflected in the work area's copy of the attribute unless a user specifically resets the attribute into the work area. However, this can potentially lead to excessive serialization and deserialization.

Excessive serialization and deserialization can result in observable performance degradation under heavy load. The deferred attribute serialization configuration property is a caching feature that reduces serialization and deserialization operations. When deferred attribute serialization is enabled in a client or server process, by selecting the Deferred Attribute Serialization field during the creation of the work area, attributes set into the work area service are not automatically serialized during the set operation. Rather, a reference to the attribute is stored in the work area. If the attribute is mutable, then changes to the object are reflected in the work area's reference to that attribute. When a get operation is performed on that attribute, the reference to that object is returned and no deserialization is performed.

Attributes are not actually serialized until the thread with which the attribute is associated makes a remote IIOP invocation. At that point the attribute is serialized and the serialized form of the attribute is cached. If the attribute is not reset into the work area, changes to the original attribute are still reflected within the attribute contained within the work area because the work area still holds a cached reference to the original object. However, if the work area has not been told that the attribute has changed by resetting the attribute into the work area, subsequent remote requests continues to use the cached serialized version of the attribute and direct changes to the mutable attribute are not propagated. This is an important distinction between enabling and not enabling the deferred attribute serialization configuration property and a user must pay close attention to this difference and how mutable objects are handled when enabling

deferred attribute serialization. The work area service releases cached references and cached serialized versions of attributes when any of the following occur:

- An attribute is reset or removed.
- The work area is explicitly completed by the application.
- Server component ends execution of the request during which the work area was begun.
- Client process which began the work area terminates.

Partition context propagation across process boundaries

Work area context automatically propagates from client to server when a client makes a remote call to a server. If a client is configured with, for example, three different work area partitions when it makes a remote call to a server, server1; the context associated with each partition on the client thread propagates to server1. If the same three partitions reside (have been created) on server1, the context is demarshalled to the appropriate partition. However, if none or only a few of the three partitions have been created on server1, only the context associated with a partition that is resident on both the client and server is demarshalled. The context associated with a partition that is not resident on server1 is still resident on server1 but will not be accessible. The context associated with partitions that are not resident on server1 must remain resident on server1 in case another remote call is made to a different server. Going one step further, if server1 makes a call to yet another server, server2 and assume server2 has created all the same partitions that the client has, server2 receives the context for the partitions that were not resident on server1. Any partitions that reside on server1 that did not reside on the client, now have its context propagated to server2.

For additional information about work area, see the com.ibm.websphere.workarea package in the Application Programming Interface (API). The generated API documentation is available in the information center table of contents from the path **Reference** → **APIs - Application Programming Interfaces**.

The Work area partition manager interface

Applications interact with the work area partition service by using the work area partition manager interface. A user can retrieve an instance of the work area partition manager interface out of naming and use the methods that are defined in the following section.

An implementation of the work area partition manager interface is bound in Java naming at `java:comp/websphere/WorkAreaPartitionManager`. This interface is responsible for creating, retrieving, and manipulating work area partitions:

```
package com.ibm.websphere.workarea;

import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.PartitionAlreadyExistsException;
import com.ibm.websphere.workarea.NoSuchPartitionException;
import java.util.Properties;

public interface WorkAreaPartitionManager {

  //Returns an instance of a work area partition for the given name, or throws an exception if the
  //partition name doesn't exists.
  public UserWorkArea getWorkAreaPartition(String partitionName) throws NoSuchPartitionException;

  //Returns a new instance of a work area partition (an implementation of the UserWorkArea interface)
  //or throws an exception if the partition name already exists.  The createWorkAreaPartition should
  //only be used within a Java 2 platform, Enterprise Edition (J2EE) client and NOT on the
  //server.  To create a work area partition on the server, use the WebSphere administrative
  //console.
  public UserWorkArea createWorkAreaPartition(String partitionName, Properties props) throws
     PartitionAlreadyExistsException, java.lang.IllegalAccessException;
  }
}
```

EJB applications can use the work area partition manager interface only within the implementation of methods in either the remote or local interface, or both; likewise, servlets can use the interface only within the service method of the HTTPServlet class. Use of work areas within any life cycle method of a servlet or enterprise bean is considered a deviation from the work area programming model and is not supported.

Programmatically creating a work area partition through the createWorkAreaPartition method is only available on the Java 2 platform, Enterprise Edition (J2EE) client. To create a work area partition on the server, use the WebSphere administrative console as described in “Configuring work area partitions” on page 2384. All partitions in a server process must be created before server startup is complete so that the work area service can register with the appropriate container collaborators. Therefore, calling the createWorkAreaPartition method in a server process after the server starts results in a java.lang.IllegalAccessException exception. The createWorkAreaPartition method can be called in a J2EE application client at any time.

Configurable Work Area Partition Properties

This section applies to the use of the createWorkAreaPartition method on the WorkAreaPartitionManager interface. As is described above, this method should only be used on a Java 2 platform, Enterprise Edition (J2EE) client. To create a partition on the server, please see Configuring work area partitions.

The ″createWorkAreaPartition″ method on the WorkAreaPartitionManager interface takes a java.util.Properties objects. This Properties object, and the properties it contains, is used to define the work area partition. Below is an example of creating a Properties object and setting a property:

Note: A more detailed example of the usage of the WorkAreaPartitionManager can be found at “Example: Using the work area partition manager” on page 2390.

```
java.util.Properties props = new java.util.Properties():
props.put("maxSendSize","12345");
```

Acceptable key/values pairs (properties) for defining a partition are as follows:

- *maxSendSize* - Indicates the maximum size (bytes) of a work area that can be sent on a remote call. Acceptable values are:
  - ″-1″ = Uses the default size of 32767.
  - ″0″ = Unlimited size, this value will not be policed which might help performance a bit depending on the number of work area an application has.
  - ″1″ = Integer.MAX_VALUE
- *maxReceiveSize* - Indicates the maximum size (bytes) of a work area that can be received. Acceptable values are:
  - ″-1″ = Uses the default size of 32767.
  - ″0″ = Unlimited size, this value will not be policed which might help performance a bit depending on the number of work area an application has.
  - ″1″ = Integer.MAX_VALUE
- *Bidirectional* - Indicates if work area context that is changed by a downstream process should be propagated back upstream to the originator of that context. For a more complete description of this property, please see the section ″Bidirectional propagation of work area context″ at “Work area partition service” on page 2384. Acceptable values are:
  - ″true″ = Context changes will be returned from a remote call.
  - ″false″ = Context changes will not be returned from a remote call.

  Note: The default setting is ″false.″

- *DeferredAttributeSerialization* - Indicates if the serialization of attribute should be optimized to occur exactly once per process. For a more complete description of this property, please see the section ″Deferred attribute serialization of work area context″ at “Work area partition service” on page 2384. Acceptable values are:
  - ″true″
    - When an attribute is set into the work area, it will not be serialized until a remote request is made.
    - If the value is unchanged by response, the serialized form will be used for subsequent requests; the live object will be retrieved via getters.
    - When requests are made during a remote request, a value is deserialized on demand exactly once. The serialized form is used for subsequent requests from this remote process on this distributed thread; subsequent requests in process for the same attribute returns the already deserialized value. There are risks with concurrency with DeferredAttributeSerialization. After serialization in a client process, updates to the attribute arel no longer reflected in the work area's copy until the value is explicitly reset through the UserWorkArea interface. Changes made to a retrieved reference in a downstream process are not propagated to subsequent downstream requests (or returned on the reply as a changed value) unless explicitly reset through the UserWorkArea interface.
  - ″false″
    - When an attribute is set into the work area, it is immediately serialized and the bytes are stored.
    - When an attribute is retrieved from the work area, it is always deserialized from stored bytes.

  Note: The default value is ″false.″
- *EnableWebServicePropagation* - Indicates if work area context must propagate on a WebService call. Acceptable values are:
  - ″true″ = Context propagates on a WebService call.
  - ″false″ = Context does not propagate on a WebService call.

  Note: The default value is ″false.″

Exceptions

The work area partition service defines the following exceptions for use with the work area partition manager interface:

PartitionAlreadyExistsException
: This exception is raised by the createWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user tries to create a work area partition with a partition name that already exists. Partition names must be unique.

NoSuchPartitionException
: This exception is raised by the getWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user requests a work area partition with a partition name that does not exist.

java.lang.IllegalAccessException
: This exception is raised by the createWorkAreaPartition method on the WorkAreaPartitionManager implementation if a user tries to create a work area partition during run time on a server process. This method can only be used on a J2EE client process. In the server process, a partition must be created using the administrative console.

For additional information about work area, see the com.ibm.websphere.workarea package in the application programming interface (API). The generated API documentation is available in the information center table of contents from the path **Reference** > **APIs - Application Programming Interfaces**.

Example: Using the work area partition manager

The example below demonstrates the use of the work area partition manager interface. The sample illustrates how to create and retrieve a work area partition programmatically`. Please note that

programmatically creating a work area partition is only available on the Java 2 platform, Enterprise Edition (J2EE) client. To create a work area partition on the server one must use the administrative console. See “Work area partition service” on page 2384 for configuration parameters available to configure a partition.

```
import com.ibm.websphere.workarea.WorkAreaPartitionManager;
import com.ibm.websphere.workarea.UserWorkArea;
import com.ibm.websphere.workarea.PartitionAlreadyExistsException;
import com.ibm.websphere.workarea.NoSuchPartitionException;
import java.lang.IllegalAccessError;
import java.util.Properties;
import javax.naming.InitialContext;

//This sample demonstrates how to retrieve an instance of the
//WorkAreaPartitionManager implementation and how to use that
//instance to create a WorkArea partition and retrieve a partition.
//NOTE: Creating a partition in the way listed below is only available
//on a J2EE client.  To create a partition on the server use the
//WebSphere administrative console.  Retrieving a WorkArea
//partition is performed in the same way on both client and server.

public class Example {

     //The name of the partition to create/retrieve
     String partitionName = "myPartitionName";
     //The name in java naming the WorkAreaPartitionManager instance is bound to
     String jndiName = "java:comp/websphere/WorkAreaPartitionManager";

     //On a J2EE client a user would create a partition as follows:
     public UserWorkArea myCreate(){
         //Variable to hold our WorkAreaPartitionManager reference
         WorkAreaPartitionManager partitionManager = null;
         //Get an instance of the WorkAreaPartitionManager implementation
         try {
             InitialContext initialContext = new InitialContext();
             partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
         } catch (Exception e) {  }

         //Set the properties to configure our WorkArea partition
         Properties props = new Properties();
         props.put("maxSendSize","12345");
         props.put("maxReceiveSize","54321");
         props.put("Bidirectional","true");
      props.put("DeferredAttributeSerialization","true");

         //Variable used to hold the newly created WorkArea Partition
         UserWorkArea myPartition = null;

         try{
             //This is the way to create a partition on the J2EE client.  Use the
             //WebSphere Administrative Console to create a WorkArea Partition
             //on the server.
             myPartition = partitionManager.createWorkAreaPartition(partitionName,props);
         }
         catch (PartitionAlreadyExistsException e){  }
         catch (IllegalAccessException e){  }

         return myPartition;
     }

     //. . . .

     //In order to retrieve a WorkArea partition at some time later or
     //from some other class, do the following (from client or server):
     public UserWorkArea myGet(){
         //Variable to hold our WorkAreaPartitionManager reference
         WorkAreaPartitionManager partitionManager = null;
         //Get an instance of the WorkAreaPartitionManager implementation
```

```
        try {
            InitialContext initialContext = new InitialContext();
            partitionManager = (WorkAreaPartitionManager) initialContext.lookup(jndiName);
        } catch (Exception e) {  }

        //Variable used to hold the retrieved WorkArea partition
        UserWorkArea myPartition = null;
        try{
            myPartition = partitionManager.getWorkAreaPartition(partitionName);
        }catch(NoSuchPartitionException e){  }

        return myPartition;
    }
}
```

Work area partition collection

Use this page to manage the work area service.

The work area partition service supports the definition of custom work area partitions.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business Process Services** → **Work area partition service**.

Name:

Specifies the name of the work area partition that is used to retrieve the partition. This name must be unique.

Description:

Specifies the description of the work area partition.

Enable service at server startup:

Specifies whether the server attempts to start the specified service when the server starts.

Bidirectional:

Permits applications to modify the context of a work area that is imported by a J2EE request; modified properties are propagated back to the requestor environment. This option is disabled by default.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area. (0 = no limit; -1 = default)

A value of 0 means there is no limit to the data sent. The default value of -1 represents 32768 bytes of data sent.

Maximum receive size:

Specifies the maximum size of data that can be received within a single work area. (0 = no limit; -1 = default)

A value of 0 means there is no limit to the data received. The default value of -1 represents 32768 bytes of data received.

Deferred attribute serialization:

Specifies whether attribute serialization is deferred until the work area is propagated on a remote invocation.

Enable Web service propagation:

Specifies whether the work area partition is propagated on Web service requests. This option is disabled by default.

Work area partition settings:

Use this page to modify the work area partition settings.

The work area partition service supports the definition of custom work area partitions.

To view this administrative console page, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business Process Services** → **Work area partition service** → ***work_area_partition_name***.

Name:

Specifies the name of the work area partition that is used to retrieve the partition. This name must be unique.

Description:

Specifies the description of the work area partition.

Enable service at server startup:

Specifies whether the server attempts to start the specified service when the server starts.

Bidirectional:

Permits applications to modify the context of a work area that is imported by a J2EE request; modified properties are propagated back to the requestor environment. This option is disabled by default.

Maximum send size:

Specifies the maximum size of data that can be sent within a single work area. (0 = no limit; -1 = default)

Data type	Integer
Units	Bytes
Default	32768
Range	-1, 0 (no limit) and 1 to 2147483647

Maximum receive size:

Specifies the maximum size of data that can be received within a single work area. (0 = no limit; -1 = default)

Data type	Integer
Units	Bytes
Default	32768
Range	-1, 0 (no limit) and 1 to 2147483647

Deferred attribute serialization:

Specifies whether attribute serialization is deferred until the work area is propagated on a remote invocation. This option is disabled by default.

Enable Web service propagation:

Specifies whether the work area partition is propagated on Web service requests. This option is disabled by default.

Accessing a user defined work area partition

About this task

The work area partition service provides a Java Naming and Directory Interface (JNDI) binding to an implementation of the work area partition manager interface under the name java:comp/websphere/WorkAreaPartitionManager. Applications that need to access their partition can perform a lookup on that JNDI name and then use the getWorkAreaPartition method on the work area partition manager, as shown in the following code example:

Example

```
import com.ibm.websphere.workarea.*;
import javax.naming.*;

public class SimpleSampleServlet {
  ...

        //Variable to hold our WorkAreaPartitionManager implementation
        WorkAreaPartitionManager partitionManager = null;
        try {
            InitialContext initialContext = new InitialContext();
            partitionManager = (WorkAreaPartitionManager)
           initialContext.lookup("java:comp/websphere/WorkAreaPartitionManager");
        } catch (Exception e) {...}

        //Variable used to hold the retrieved WorkArea Partition
        UserWorkArea myPartition = null;
        try{
            myPartition = partitionManager.getWorkAreaPartition(partitionName);
        }catch(NoSuchPartitionException e){...}
}
```

What to do next

The next step is to use the begin method to create a new work area and associate it with the calling thread, as described in the topic Beginning a new work area.

Propagating work area context over Web services

WebSphere Application Server Version 6.1 introduces the option to propagate work area context on a Web service call. Prior to WebSphere Application Server Version 6.1, work area context was only propagated over RMI/IIOP calls. The work area application programming interfaces (APIs) have not changed to implement this propagation. You can use the work area APIs as they have in the past and as outlined in the work area documentation. However, by default, work area context is not propagated on a Web service call, you must enable this option.

1. Enable a server to propagate work area context on a Web service call.
   a. Start the administrative console.
   b. Select **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **Business Process Services** .
      - To enable the work area service, (the UserWorkArea partition) to propagate its context on a Web service call:
        – Select **Work area service**.

- To enable an individual partition to propagate its context on a Web service call:
  - Select **Work area partition service**.
  - Select a partition.

c. Check the EnableWebServicePropagation field to enable Web service propagation.

d. Save the new configuration and restart the server to apply the new configuration.

2. Enable a client to propagate work area context on a Web service call:

   Note: The steps below are for the work area service (the UserWorkArea partition). For user defined partitions the EnableWebServicePropagation property must be set when creating a partition on the client, see “The Work area partition manager interface” on page 2388.

   a. Set the property com.ibm.websphere.workarea.EnableWebServicePropagation to true when invoking the launchClient script found in the $WAS_HOME/bin directory. For example, to set this property to true, add the following system properties to the launchClient invocation as needed: -CCDcom.ibm.websphere.workarea.EnableWebServicePropagation=true

   b. Set the property com.ibm.websphere.workarea.EnableWebServicePropagation in a property file that is used by the launchClient script. See “Running application clients” on page 203for additional information.

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic describes the conventions in use for WebSphere Application Server.

Default product locations - IBM i

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server products or components require multiple locations.

app_client_root
The default installation root directory for the Java EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V7/client directory.

app_client_user_data_root
The default Java EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V7/client directory.

app_client_profile_root
The default Java EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V7/client/profiles/*profile_name* directory.

app_server_root
The default installation root directory for WebSphere Application Server - Express is the /QIBM/ProdData/WebSphere/AppServer/V7/Express directory.

cip_app_server_root
The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_profile_root
The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_user_data_root
The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V7/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

if_root This directory represents the root directory of the IBM WebSphere Installation Factory. Because you can download and unpack the Installation Factory to any directory on the file system to which you have write access, this directory's location varies by user. The Installation Factory is an Eclipse-based tool which creates installation packages for installing WebSphere Application Server in a reliable and repeatable way, tailored to your specific needs.

iip_root
This directory represents the root directory of an *integrated installation package* (IIP) produced by the IBM WebSphere Installation Factory. Because you can create and save an IIP to any directory on the file system to which you have write access, this directory's location varies by user. An IIP is an aggregated installation package created with the Installation Factory that can include one or more generally available installation packages, one or more customized installation packages (CIPs), and other user-specified files and directories.

java_home

The following directories are the root directories for all supported Java Virtual Machines (JVMs).

JVM	Directory
Classic JVM	/QIBM/ProdData/Java400/jdk6
32–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/32bit
64–bit IBM Technology for Java	/QOpenSys/QIBM/ProdData/JavaVM/jdk60/64bit

plugins_profile_root

The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver/profiles/*profile_name* directory.

plugins_root

The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V7/webserver directory.

plugins_user_data_root

The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V7/webserver directory.

product_library
product_lib

This is the product library for the installed product. The product library for each Version 7.0 installation on the system contains the program and service program objects (similar to .exe, .dll, .so objects) for the installed product. The product library name is QWAS7*x* (where *x* is A, B, C, and so on). The product library for the first WebSphere Application Server Version 7.0 product installed on the system is `QWAS7A`. The *app_server_root*/properties/product.properties file contains the value for the product library of the installation, `was.install.library`, and is located under the *app_server_root* directory.

profile_root

The default directory for a profile named *profile_name* for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express/profiles/*profile_name* directory.

shared_product_library

The shared product library, which contains all of the objects shared by all installations on the system, is `QWAS7`. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

updi_root

The default installation root directory for the Update Installer for WebSphere Software is the /QIBM/ProdData/WebSphere/UpdateInstaller/V7/updi directory.

user_data_root

The default user data directory for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V7/Express directory.

The profiles and profileRegistry subdirectories are created under this directory when you install the product.

web_server_root

The default web server path is /www/*web_server_name*.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.